Busca Digital
Tries e Árvore Patricia

Prof.a Bárbara Quintela
Prof. Jose J. Camata
Prof. Marcelo Caniato
camata@ice.ufjf.br
marcelo.caniato@ice.ufjf.br
barbara@ice.ufjf.br

Busca Digital

- Um problema:
  - Armazenar um texto literário em um computador para, em seguida, tentar localizar frases.

Busca Digital

- Considerando os problemas de busca vistos até aqui, admite-se que exista:
  - um conjunto de chaves $S = \{S_0, S_1, ..., S_n\}$
  - e um valor *x* que corresponde a uma chave que deseja localizar em S.
    - Cada chave S_i é tratado como elemento indivisível
    - Cada frase passível de buscar seria uma chave S_i.
    - Chaves podem assumir tamanhos diversos.

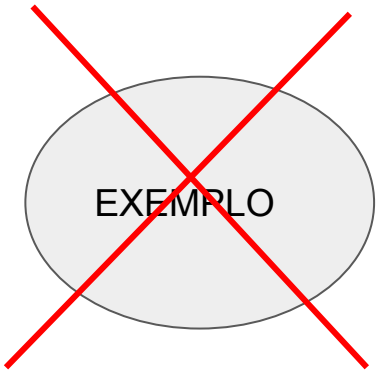
Busca Digital

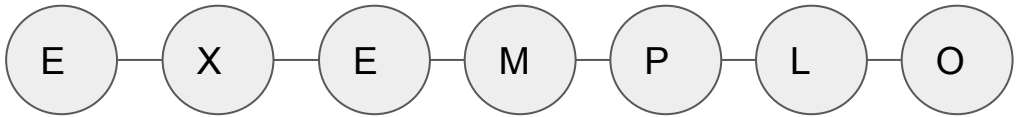
- Considerando os problemas de busca vistos até aqui, admite-se que exista:
  - um conjunto de chaves $S = \{S_0, S_1, ..., S_n\}$
  - e um valor *x* que corresponde a uma chave que deseja localizar em S.
    - Cada chave S_i é tratado como elemento indivisível
    - Cada frase passível de buscar seria uma chave S_i.
    - Chaves podem assumir tamanhos diversos.

- **E se as chaves excederem o espaço reservado?**
  - Chaves dinâmicas

Busca Digital

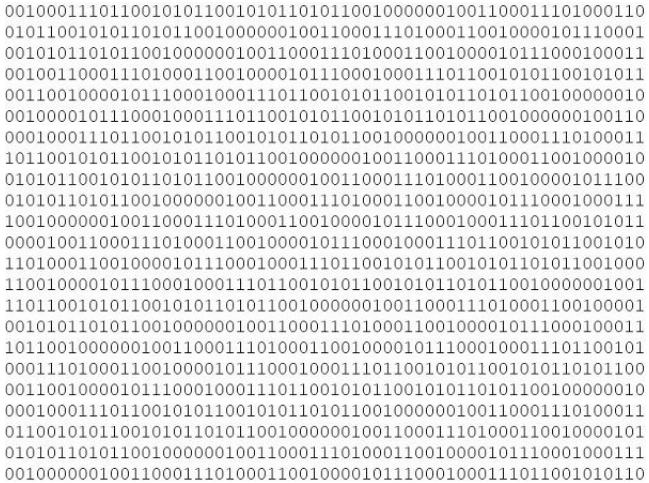
- Na busca digital, a chave não é tratada como elemento único, indivisível.
  - Cada chave é constituída por um conjunto de **caracteres** ou **dígitos** definidos em um alfabeto apropriado.





Busca Digital

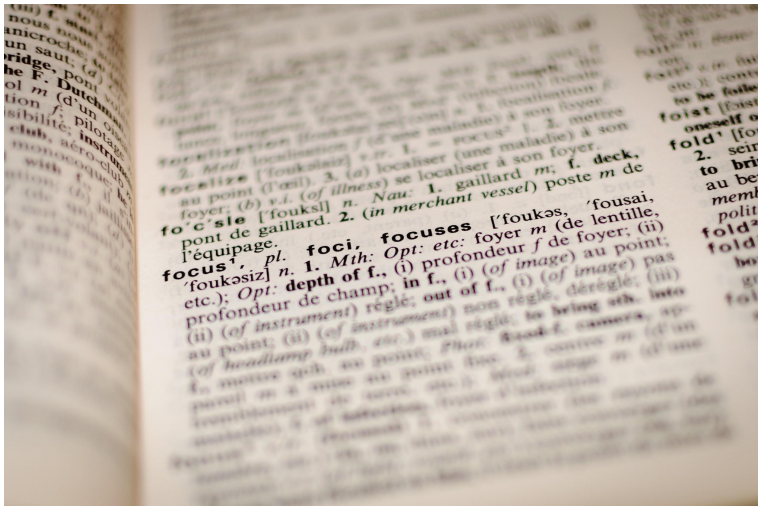
- Em vez de se **comparar a chave** procurada com as chaves do conjunto armazenado, a comparação é efetuada, individualmente, entre os dígitos que compõem as chaves, dígito a dígito.



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,

Busca Digital

- O método de pesquisa digital é análogo à pesquisa manual em dicionários:
  - com a primeira letra da palavra são determinadas todas as páginas que contêm as palavras iniciadas por aquela letra e assim por diante



Árvore Trie

- Tries são árvores de busca digital em que **toda chave está numa folha**.
- Definida em 1960 por Edward Fredkin.
- Vêm de Re**trie**val (Relacionado à Recuperação de Informações)

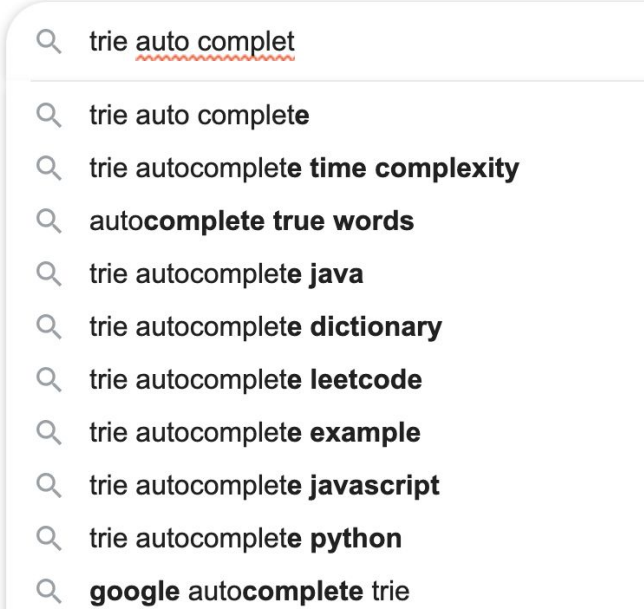
- Para distinção com **tree** pronuncia-se ***try*** **(trai)**

Árvore Trie

- **Ideia geral:** usar partes das CHAVES como caminho de busca.
  - Cada nó contém informações sobre um ou mais símbolos do alfabeto utilizado.
  - O Alfabeto pode abranger: {0,1} , {A,B,C,D...} ou {0,1,2,3,4...} e mais o caractere nulo (ou branco).

Árvores Trie

- As *Tries* são boas para suportar tarefas de tratamento lexicográfico, tais como:
  - manuseamento de dicionários;
  - pesquisas em textos de grande dimensão;
  - construção de índices de documentos;
  - expressões regulares (padrões de pesquisa).



Tries: Definições

- $S = \{s_1, \ldots, s_n\}$ é o conjunto de **chaves** a serem indexadas
- Cada chave s_i é formada por uma sequência de elementos d_j denominados **dígitos**
- Supõe-se que existe, em S, um total de **m dígitos distintos**, que compõe o **alfabeto de S**
- Os dígitos do alfabeto admitem ordenação, tal que $d_1 < \ldots < d_m$
- Os p primeiros dígitos de uma chave compõem o **prefixo de tamanho p** da chave

Tries: Definições

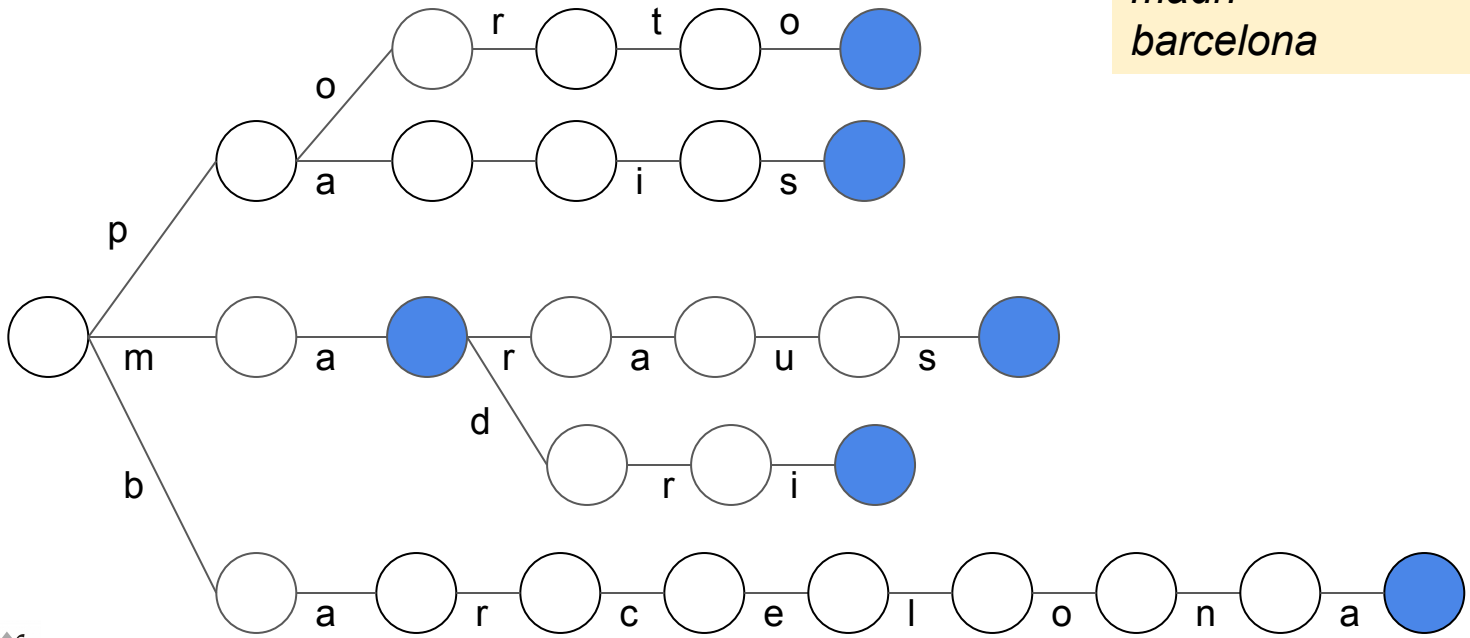
- Uma árvore digital para S é uma árvore m-ária T, não vazia, tal que:
  - Se um nó *v* é o *j*-ézimo filho de seu pai, então v corresponde ao dígito d_j do alfabeto *S*,
  - Para cada nó *v*, a sequência de dígitos definida pelo caminho desde a raiz de *T* até v corresponde a um prefixo de alguma chave de *S*.
- Decorre dessa definição que a raiz de uma trie sempre existe e não corresponde a qualquer dígito do alfabeto.

Exemplo

- S = {*porto, paris, manaus, ma, madri, barcelona*}
- *Alfabeto de S E= {a,b,c,d,..., x,z}*

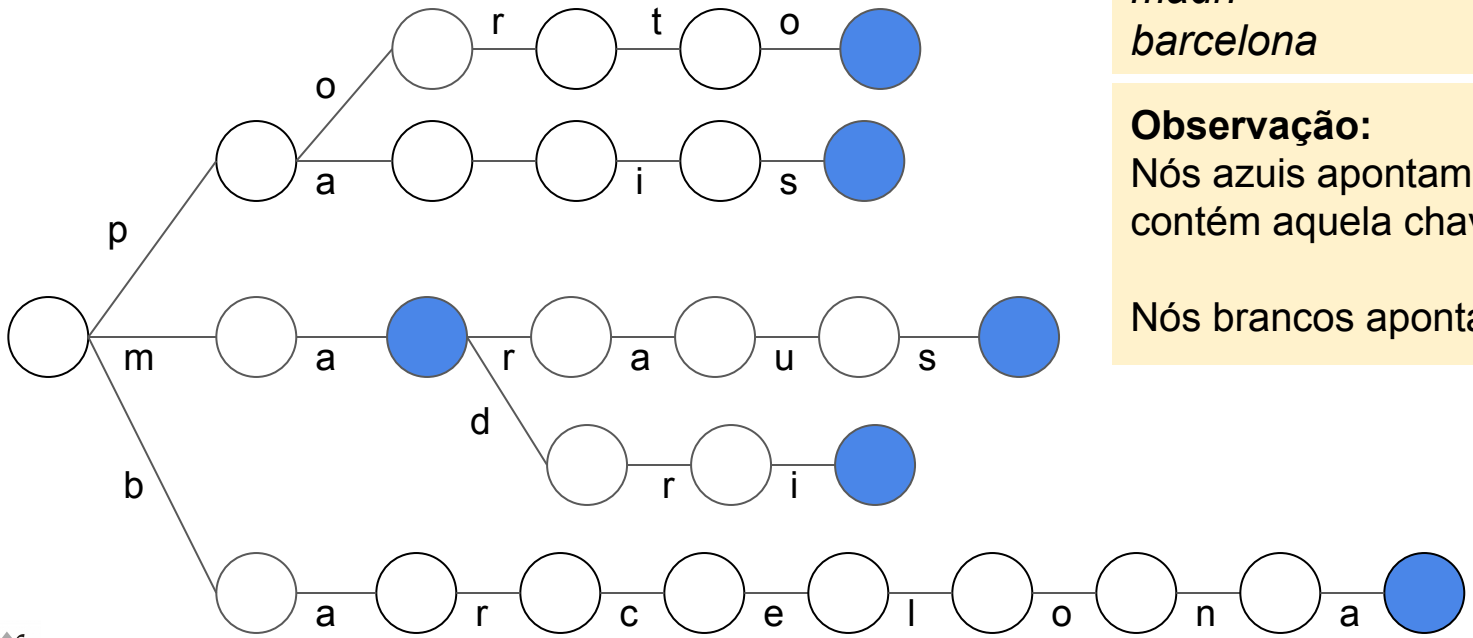
Chaves Indexadas:

porto
paris
manaus
ma
madri
barcelona



Exemplo

- S = {*porto, paris, manaus, ma, madri, barcelona*}
- *Alfabeto de S E= {a,b,c,d,..., x,z}*



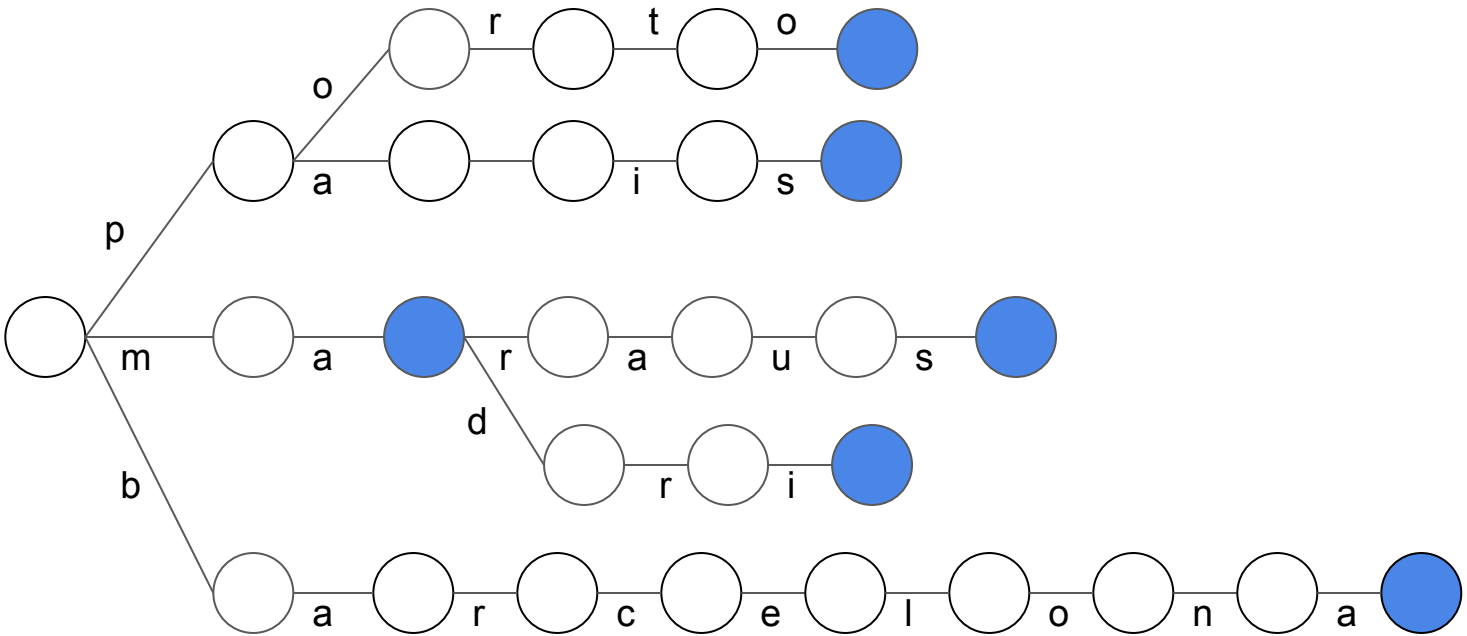
Chaves Indexadas:

porto
paris
manaus
ma
madri
barcelona

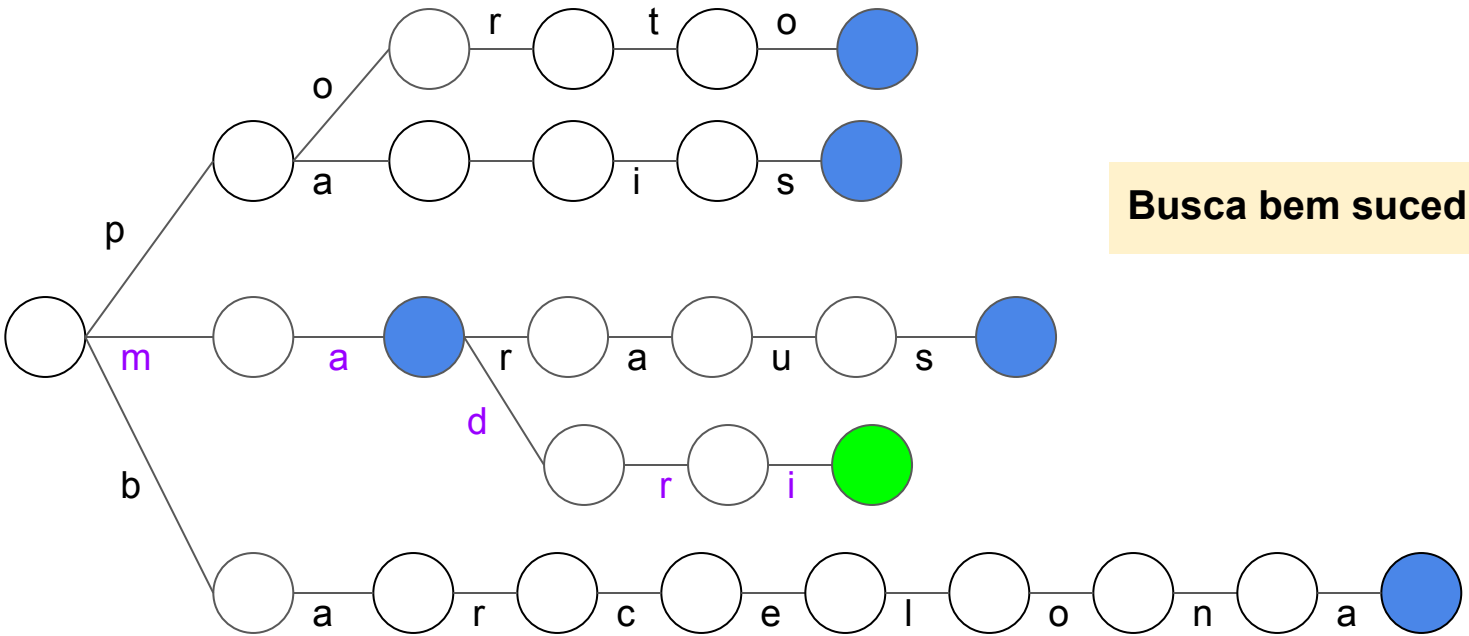
Observação:
Nós azuis apontam para registro que contém aquela chave

Nós brancos apontam para NULL

Buscar a chave madri

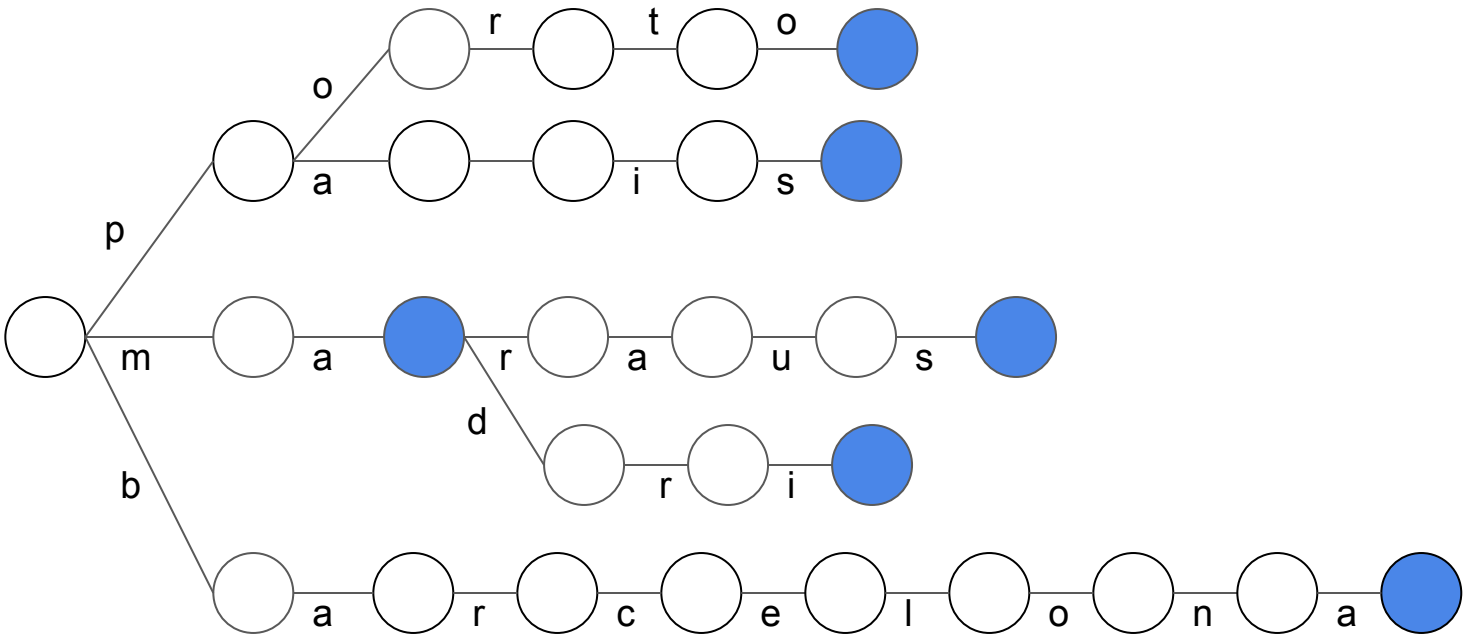


Buscar a chave madri

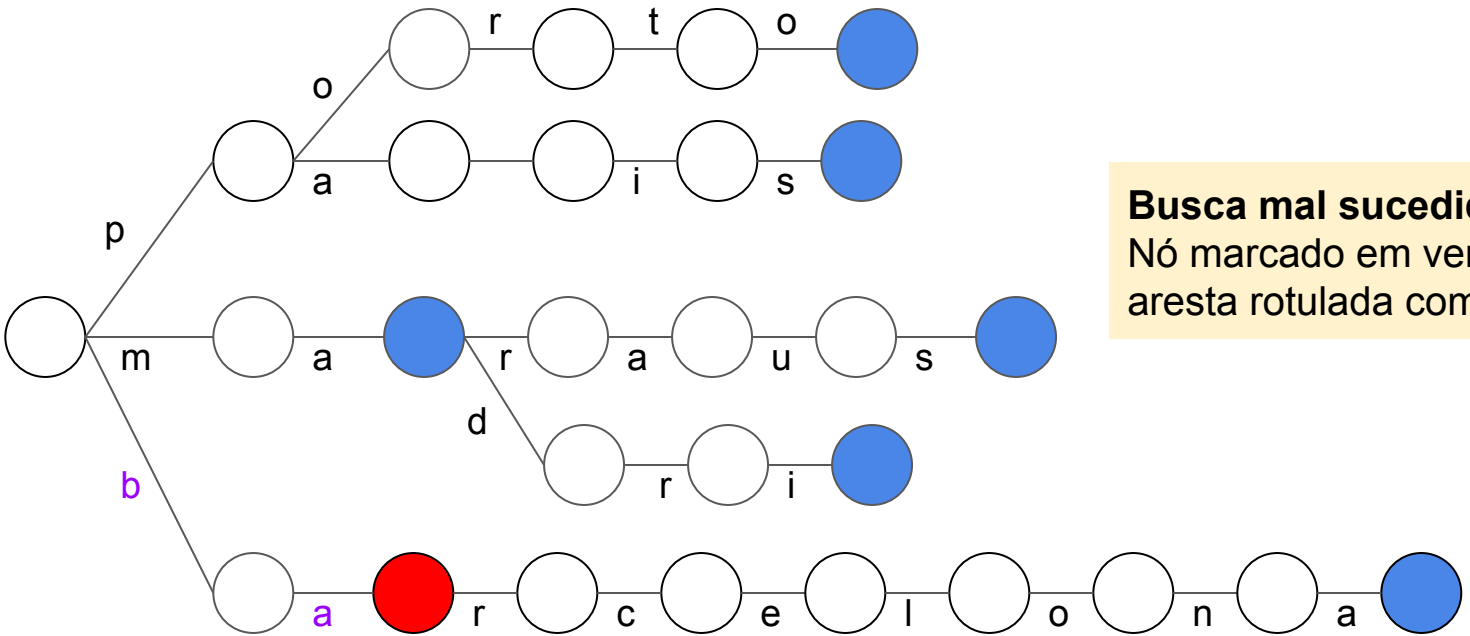


Busca bem sucedida!

Buscar a chave bahia



Buscar a chave bahia



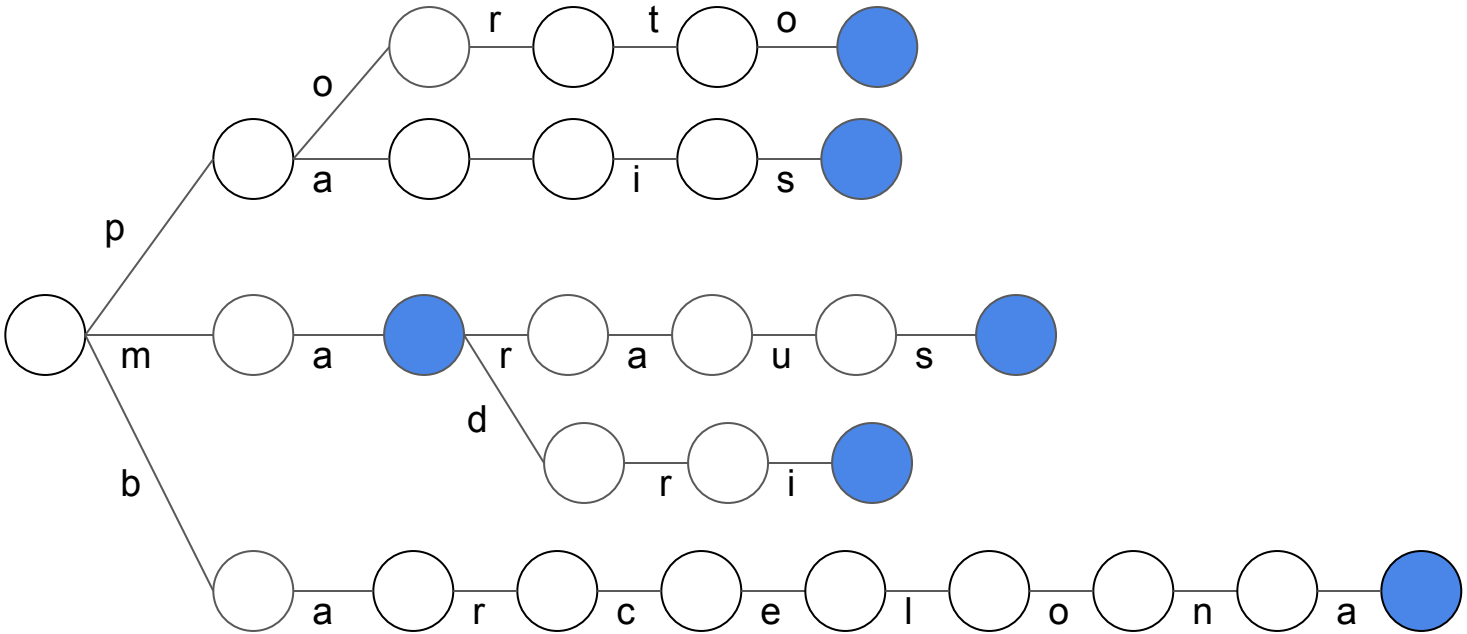
Busca mal sucedida!
Nó marcado em vermelho não tem uma aresta rotulada com ‘h’

Inserção de chaves

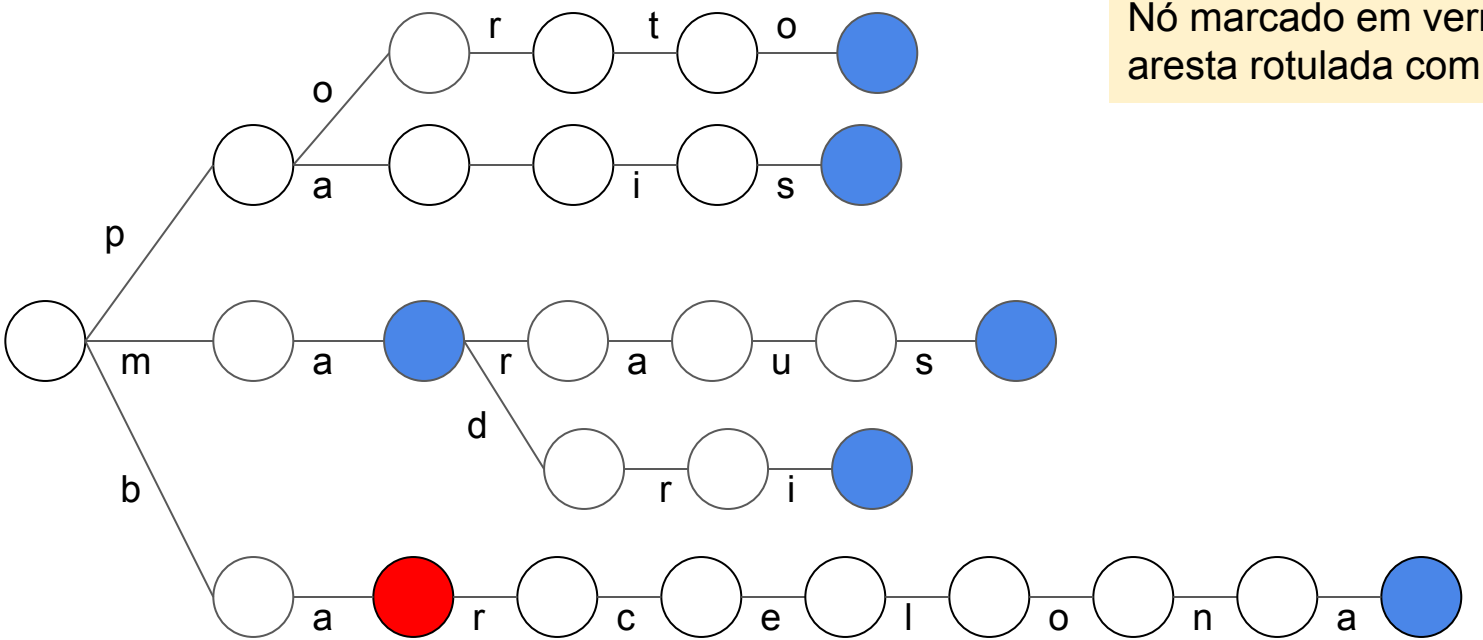
Passos:

1. Faz-se uma busca pela palavra a ser inserida. Se ela já existir na TRIE nada é feito.
2. Caso contrário,é recuperado o último nó n da maior substring da palavra a ser inserida
3. O restante dos caracteres da chave são adicionados na TRIE a partir do nó n.

Inserindo chave bahia

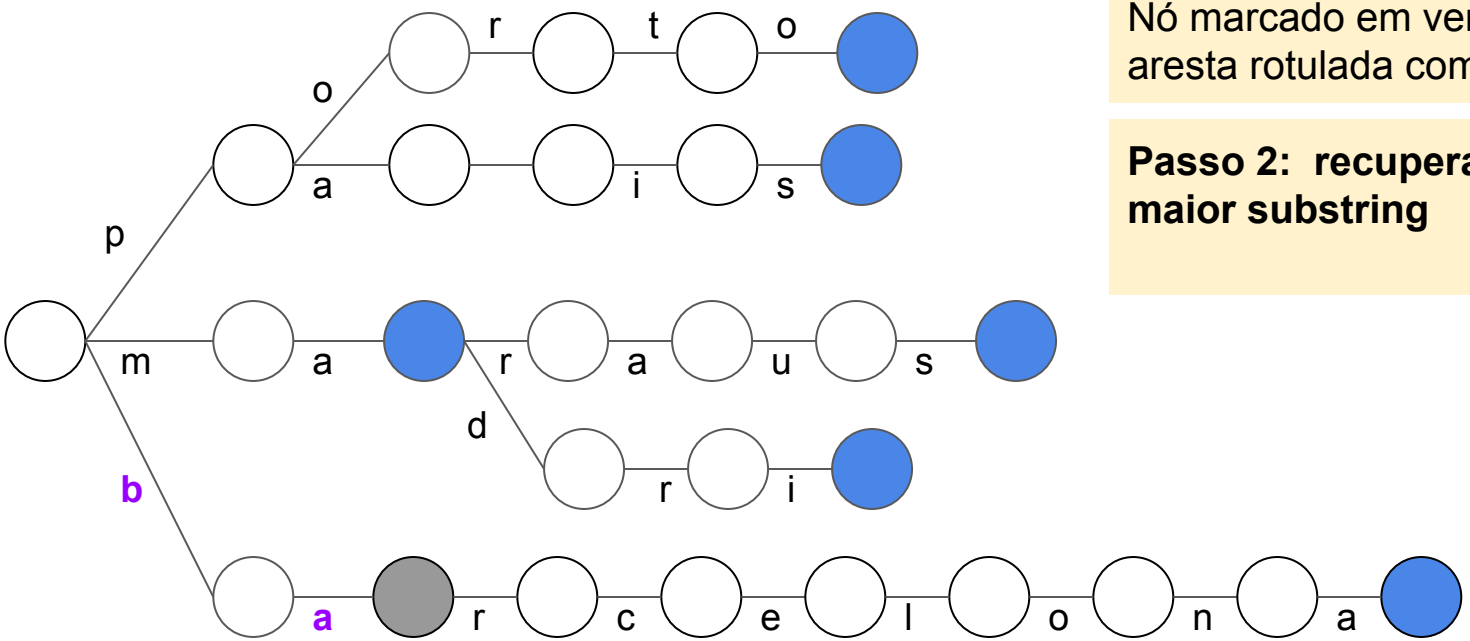


Inserindo chave bahia



Passo 1: Busca mal sucedida!
Nó marcado em vermelho não tem uma aresta rotulada com ‘h’

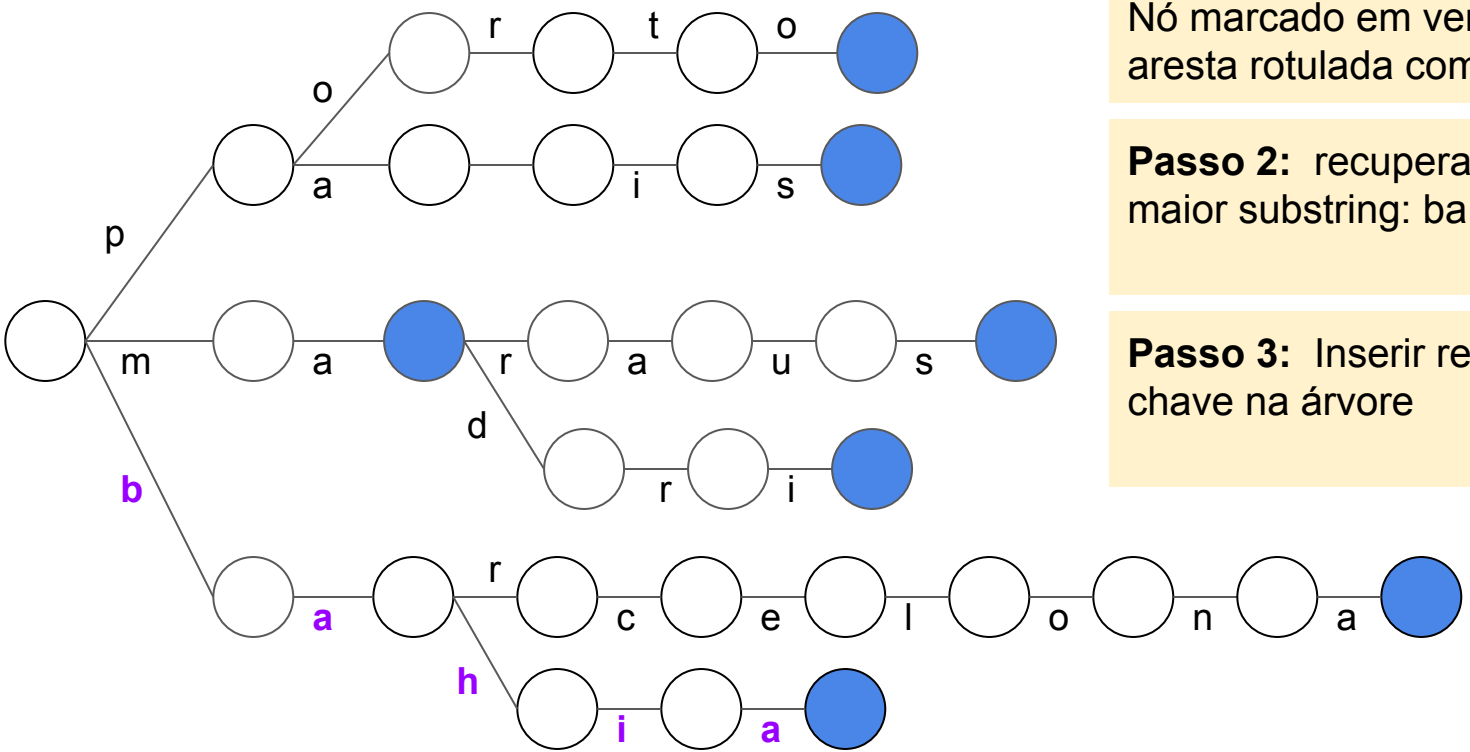
Inserindo chave bahia



Passo 1: Busca mal sucedida!
Nó marcado em vermelho não tem uma aresta rotulada com ‘h’

Passo 2: recuperado o último nó da maior substring

Inserindo chave bahia



Passo 1: Busca mal sucedida!
Nó marcado em vermelho não tem uma aresta rotulada com ‘h’

Passo 2: recuperado o último nó da maior substring: ba

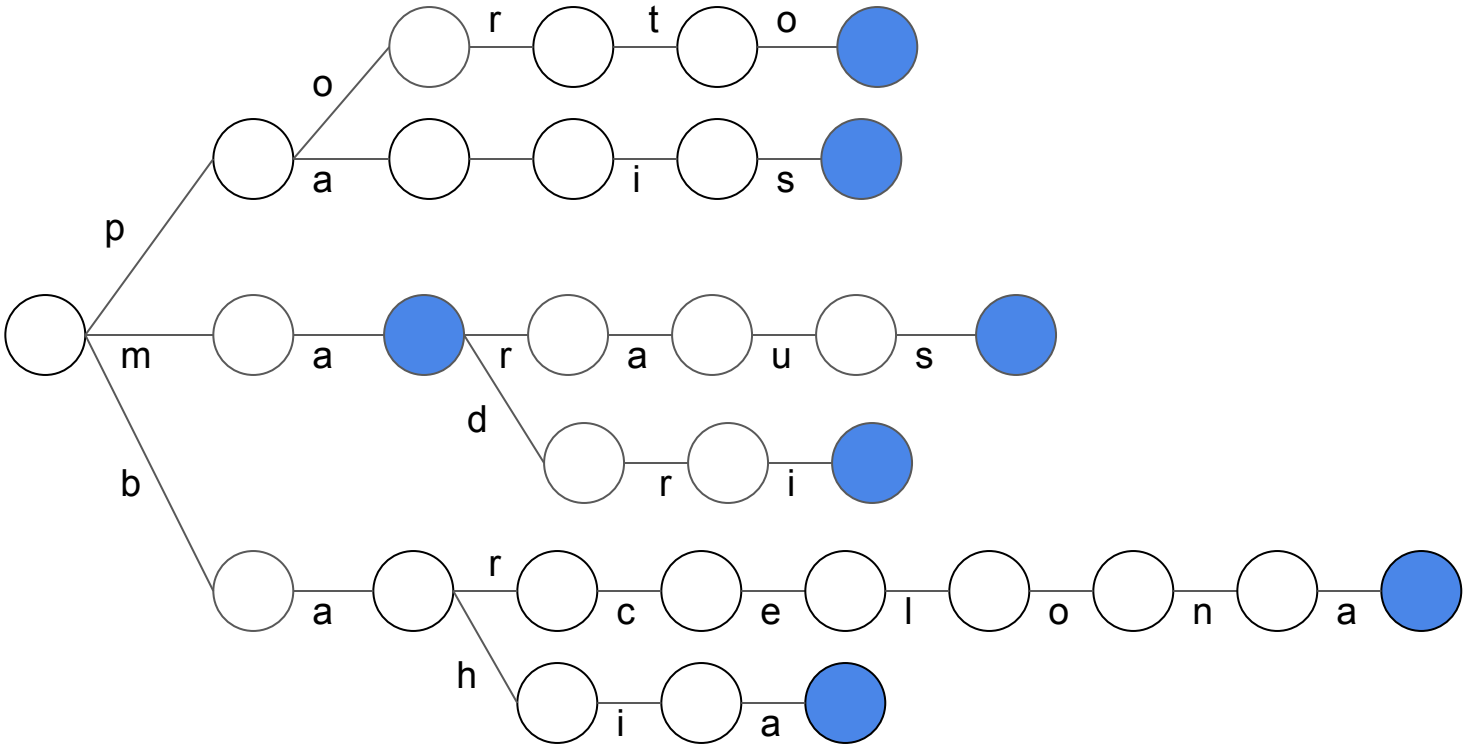
Passo 3: Inserir restante dos dígitos da chave na árvore

Remoção de chaves

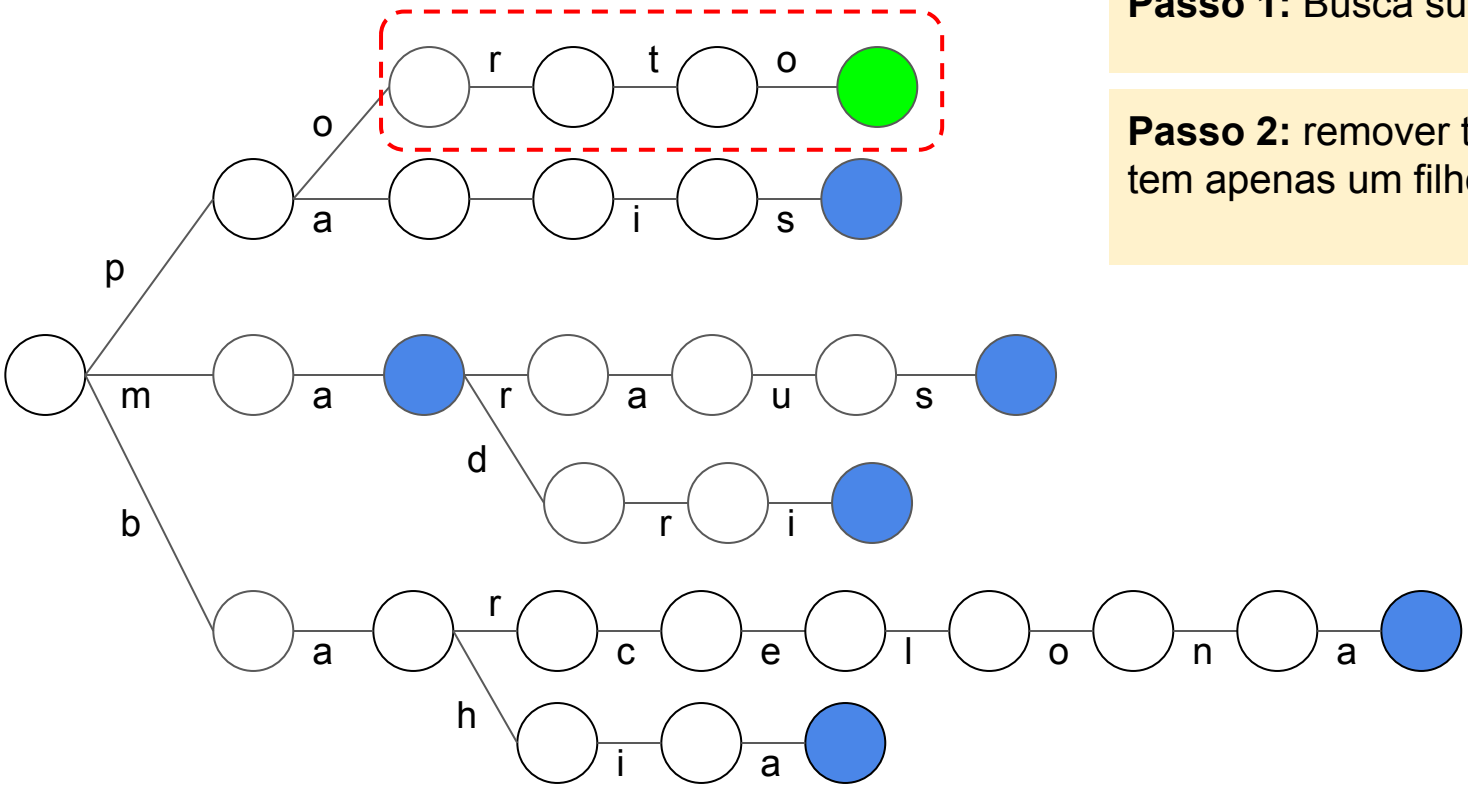
Passos:

1. Faz-se uma busca pela palavra a ser removida.
2. A partir da folha, são removidos todos os nós que têm apenas um filho (bottom-up).

Removendo chave porto



Removendo chave porto



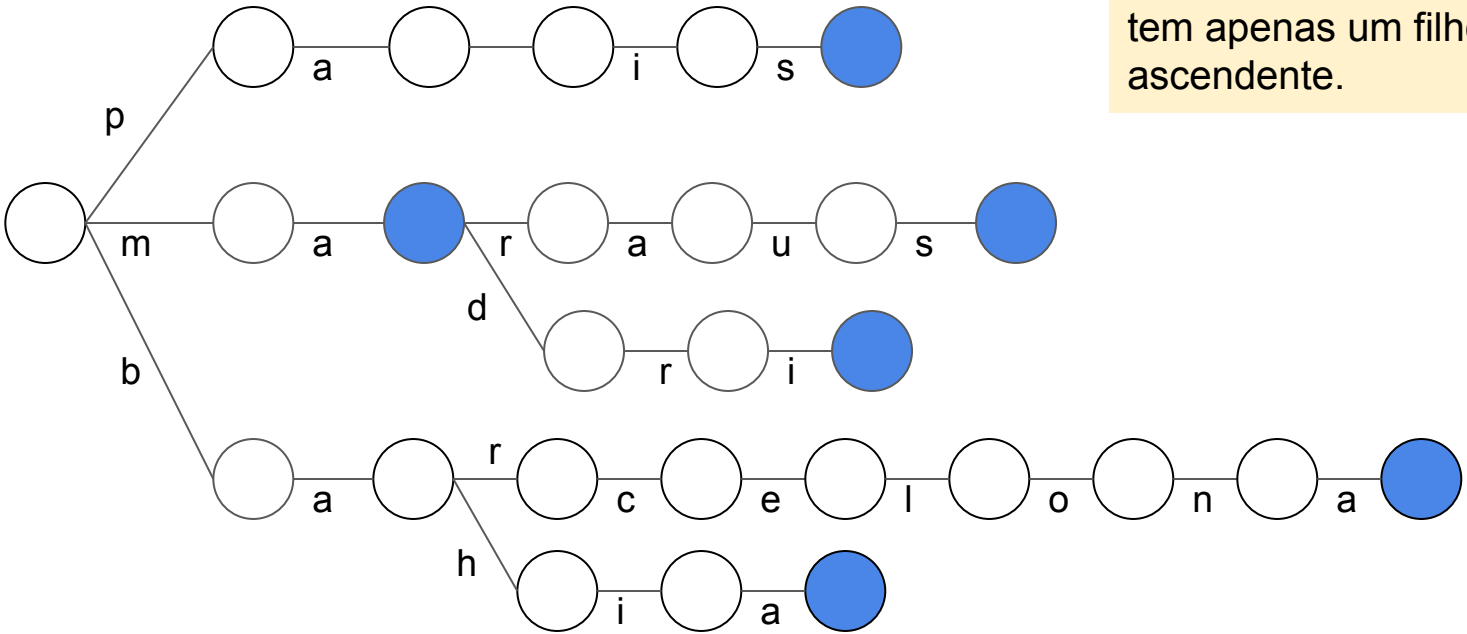
Passo 1: Busca sucedida!

Passo 2: remover todos os nós que tem apenas um filho

Removendo chave porto

Passo 1: Busca sucedida!

Passo 2: remover todos os nós que tem apenas um filho no caminho ascendente.



Algumas Considerações

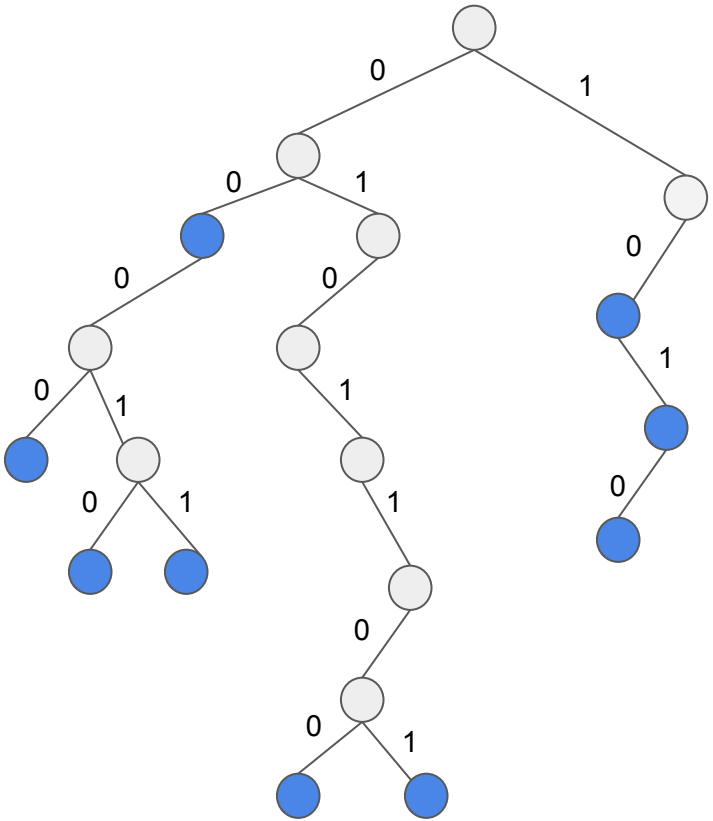
- Consome muito espaço de memória
  - Para uma árvore de ordem ***m*** com ***n*** chaves de tamanho ***t*** com prefixos distintos, o espaço necessário seria **O(mnt)**.
- Árvores com muitos zigue-zagues quase sempre são ineficientes (um zigue-zague de uma árvore T é uma subárvore parcial cujos nós possuem apenas um único filho em T)

Árvores Digitais Binárias

- É simplesmente o caso binário de uma árvore digital, onde m = 2
  - Alfabeto é representado por {0,1}
  - Vista como uma árvore binária
    - A Seleção do filho esquerdo é interpretada como o dígito 0 e o direito como 1.
- Vantagens:
  - Número de ponteiros NULL é menor (ocupa menos espaço)
  - Chaves podem ser facilmente convertidas em binário
- Desvantagens:
  - Zigue-zagues ainda podem ocorrer

Árvore Digital Binária
Exemplo

Chaves Indexadas:
00
0000
00010
00011
0101100
0101101
10
101
1010



Exercício

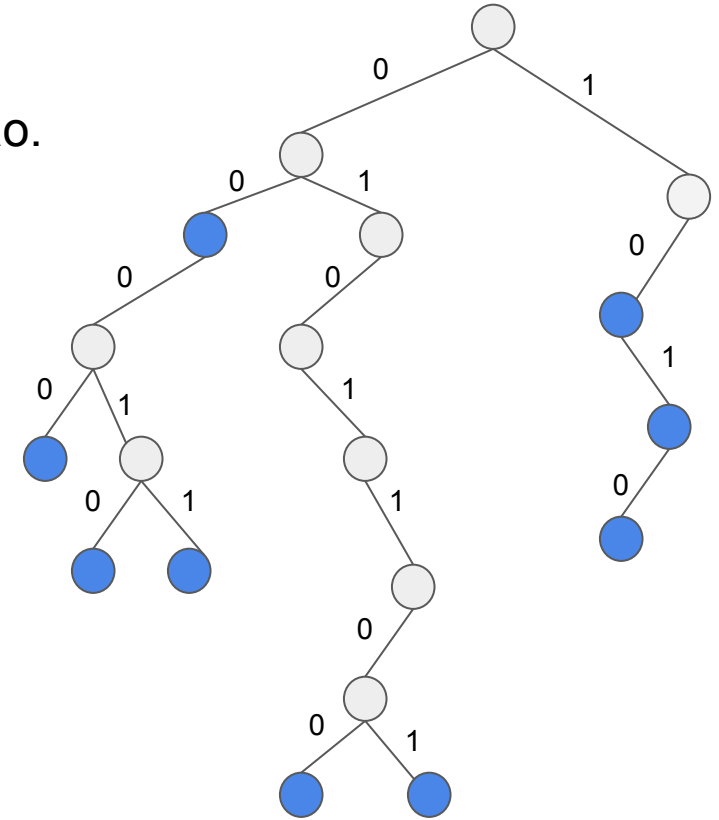
1. Mostre a trie binária resultante da inserção das chaves `00100 00001 10110 11100 11101`

Visualização da Trie

Árvore Binária de Prefixo

- Ao analisar as chaves, verificamos que algumas são prefixos de outras na coleção.
- Do exemplo ao lado:

00	010110	10
0000	010111	101
00010		1010
00011		



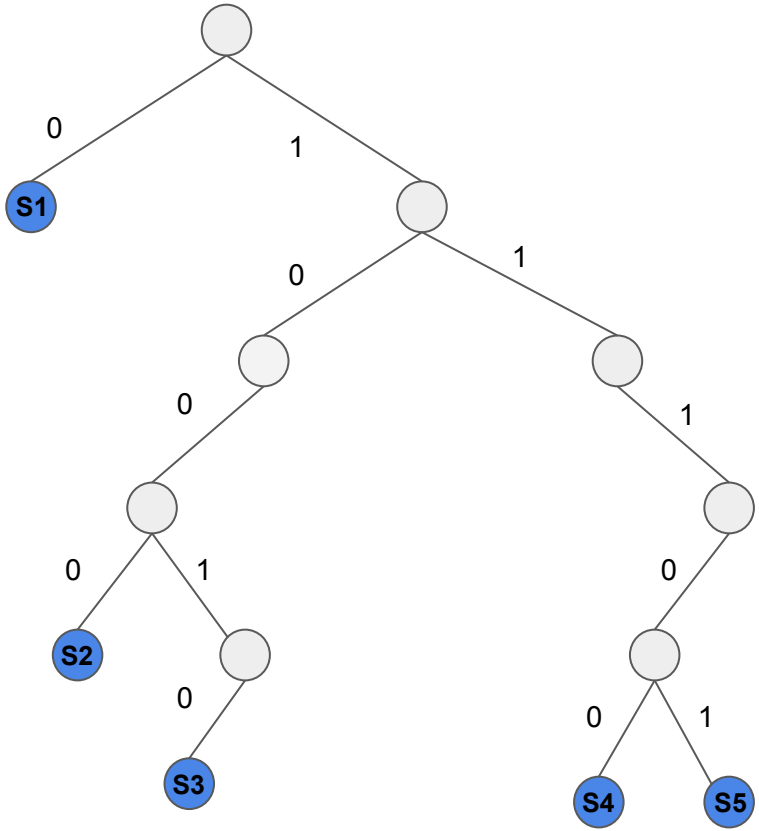
- Frequentemente, para melhor manipular a estrutura, deseja-se que tal situação não aconteça
- Assim, uma árvore binária de prefixo é uma árvore digital binária tal que nenhum código seja prefixo do outro.

Árvore Digital Binária Exemplo

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

É Árvore de Prefixo, pois nenhuma chave é prefixo de outra. Todas as chaves estão nas folhas

Propriedade interessante:
Todas as chaves estão nas folhas



Árvore PATRICIA

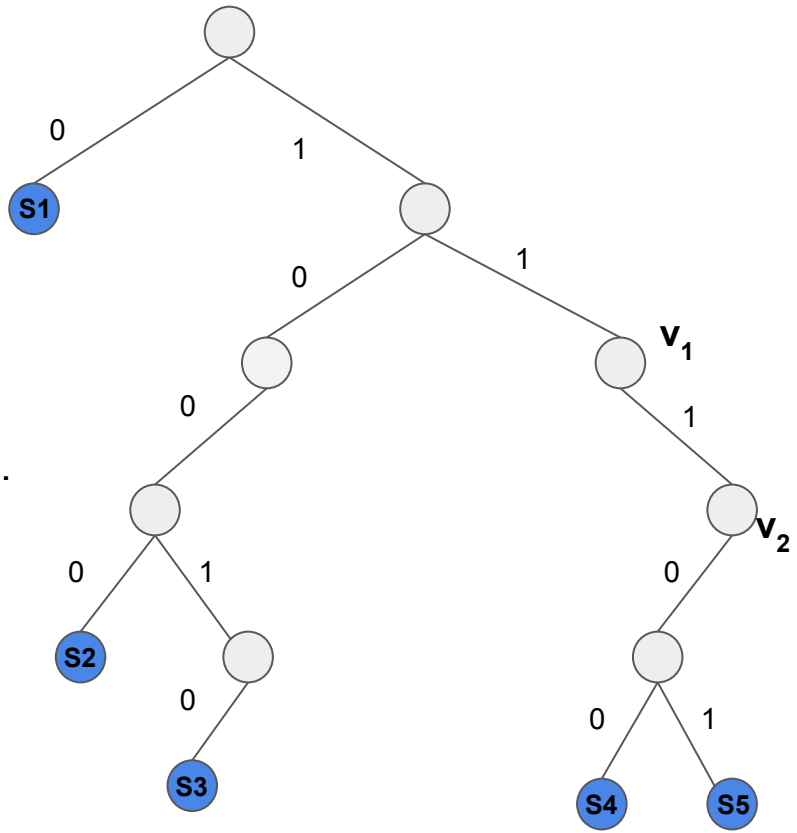
- **Implementação eficiente de Árvore Digital Binária**
- **PATRICIA**: abreviatura de
  - **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric (Algoritmo Prático para Recuperar Informação Codificada em Alfanumérico)
- Construída a partir de árvore binária de prefixo.

Árvore PATRICIA

- Nas Árvores Patricia
  - Nós internos são aproveitados para armazenar chaves
  - Longos caminhos são evitados
- S = $\{s_1, s_2, \ldots, s_n\}$ são as chaves indexadas pela árvore
- Nenhuma chave s_i é prefixo de outra.

Motivação: Buscar por S4 (11100) em uma Árvore Digital de Busca

- A certo ponto do caminho, a busca entra em um zigue-zague
- Em cada nó do zigue-zague, há sempre uma única opção de caminho.
- Ideia:
  - Ao chegar em v_1 pular para o quinto dígito.
  - É preciso criar um rótulo em v_1 com essa informação.



Chaves Indexadas:

S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

Construção da Árvore Patrícia

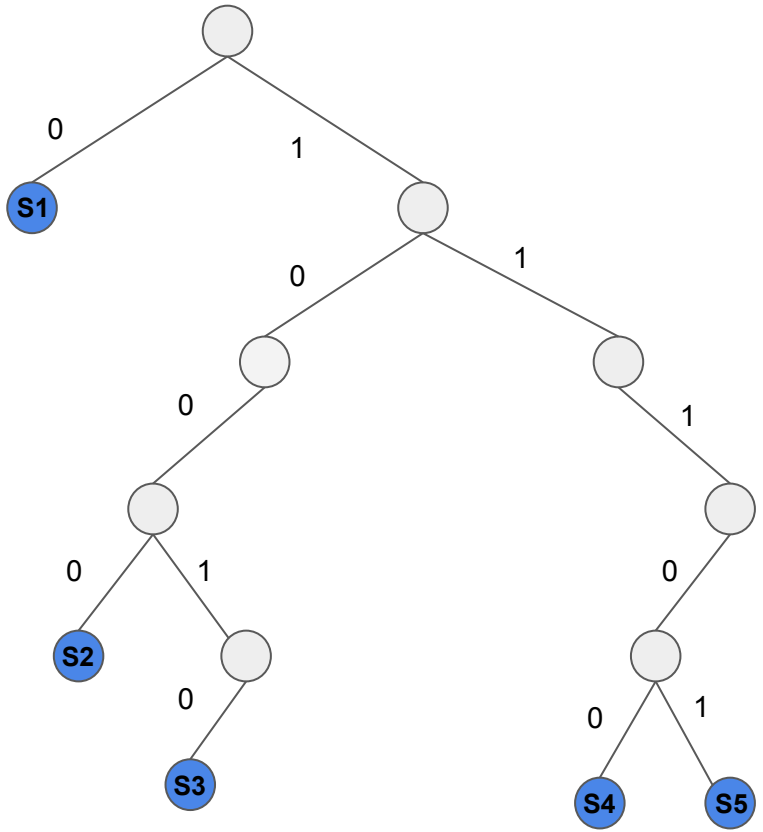
- Seja H a Árvore de Prefixos original
- Pode-se construir a Árvore Patrícia T da seguinte forma:
  - Para cada zigue-zague v_1, ..., v_k em H:
    - Se v_k é um nó folha referente à chave s_i ,
      - compactar v_1, ..., v_k em v_1 e definir $r(v_i) = s_i$.
    - Caso contrário, v_k possui o filho w.
      - Compactar v_1, ..., v_k, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$
  - Se algum vértice v de T permaneceu sem rótulo
    - definir rótulo $r(v) = nível_H(v)$

Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

Para cada zigue-zague v_1, ..., v_k em H:

- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir $r(v_i) = s_i$.
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

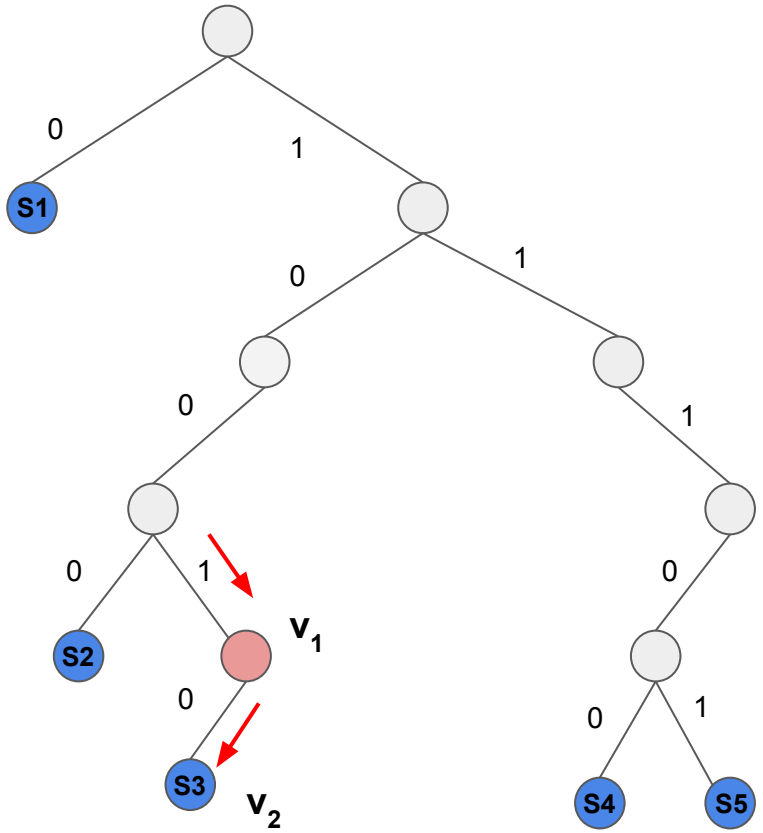


Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

Para cada zigue-zague v_1, ..., v_k em H:

- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir $r(v_i) = s_i$.
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

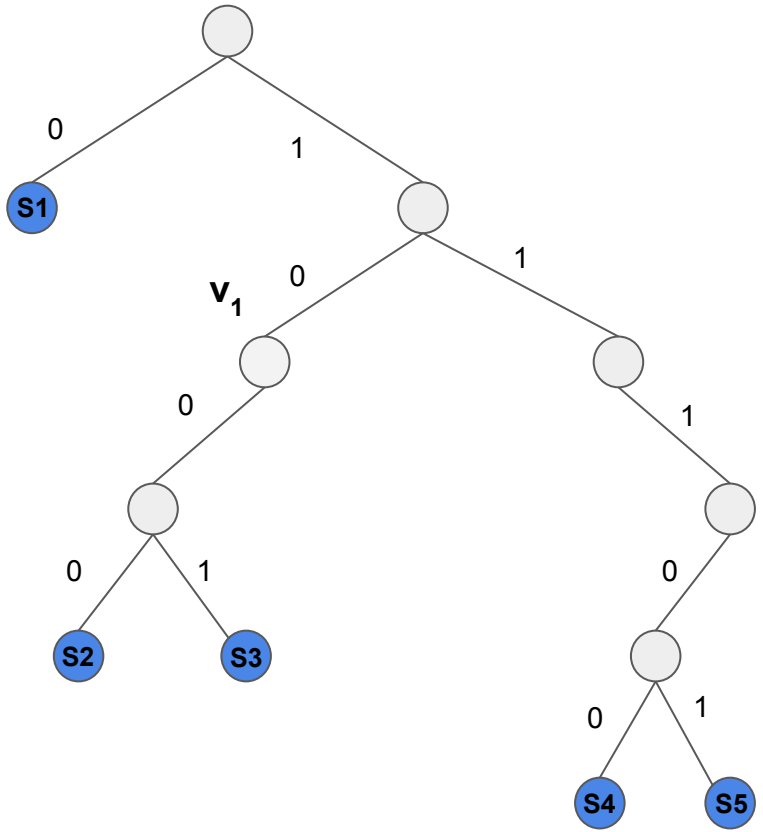


Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

Para cada zigue-zague v_1, ..., v_k em H:

- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir $r(v_i) = s_i$.
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir $r(v_1) = nivel_H(v_1) + k$

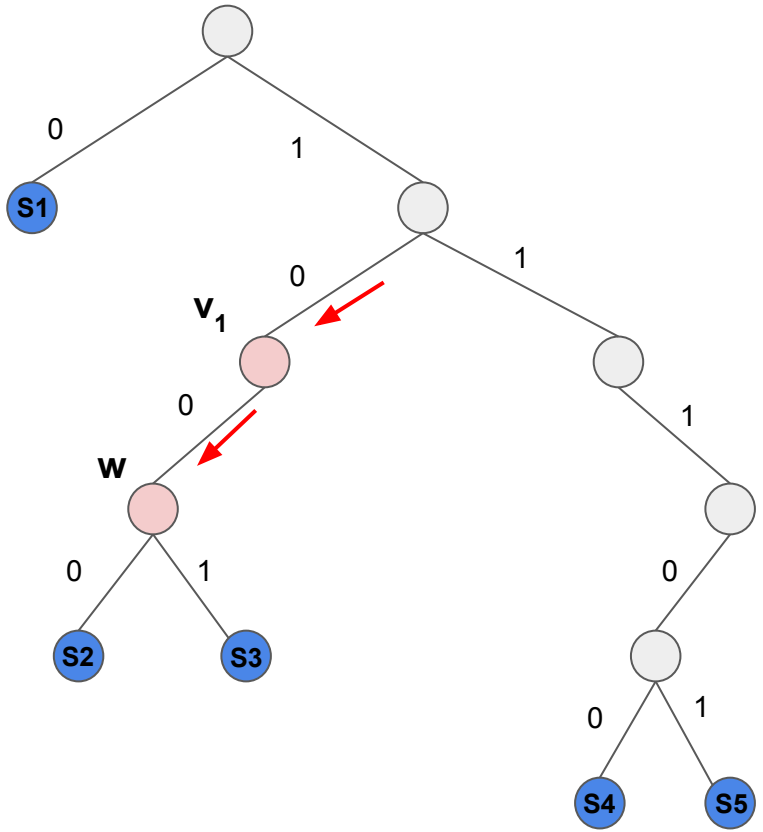


Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

nível(v_1) = 3

Compactar v_1 e w



Para cada zigue-zague v_1, ..., v_k em H:

- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir $r(v_i) = s_i$.
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir $r(v_1)$ = nível(v_1) + k

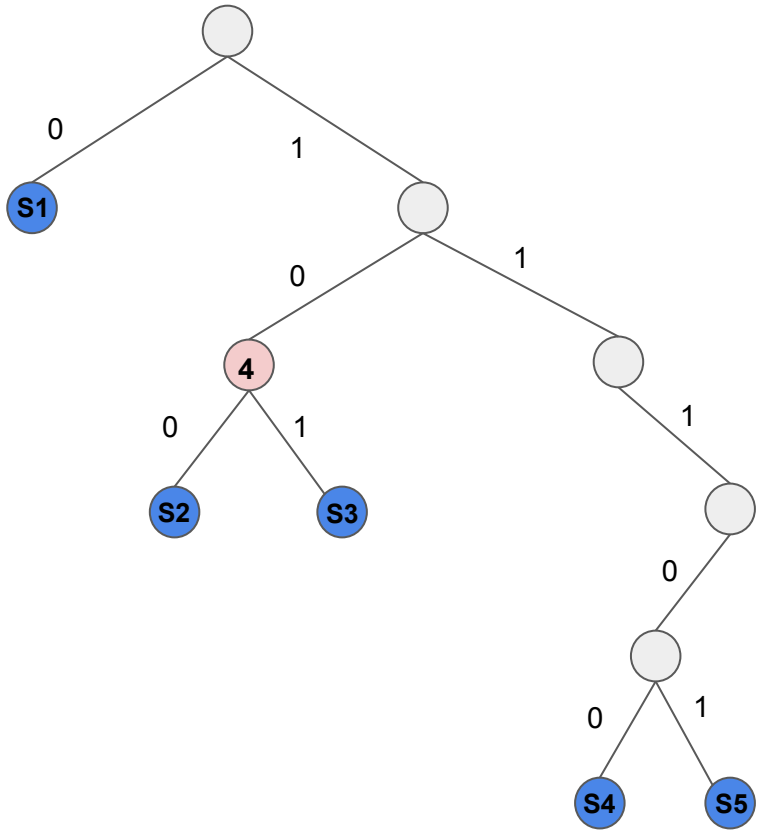
Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

nível(v_1) = 3

Compactar v_1 e w

Redefinir r(v_1)



Para cada zigue-zague v_1, ..., v_k em H:

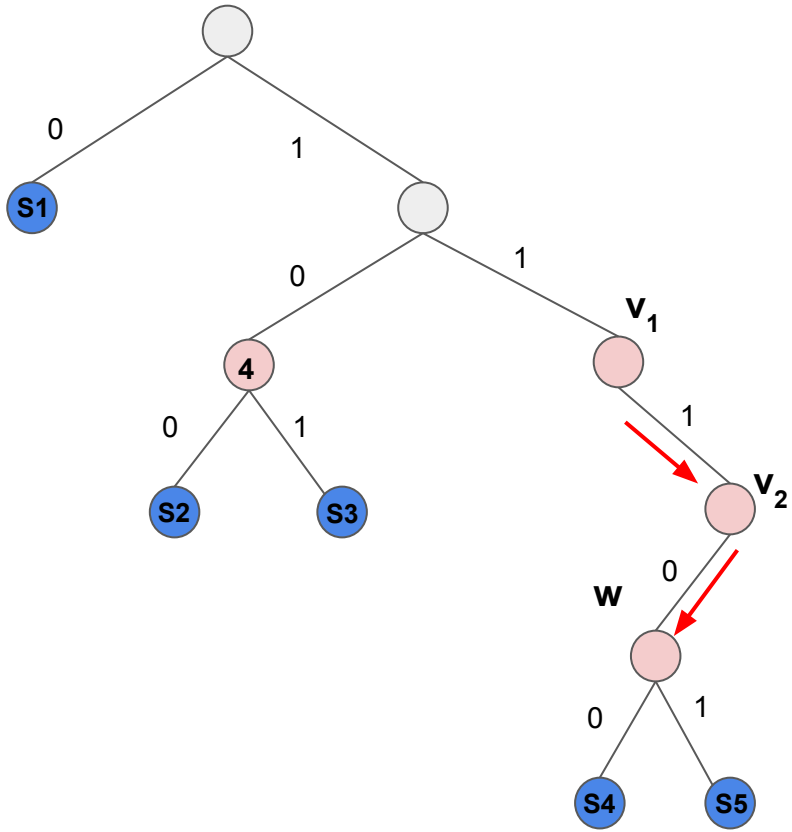
- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir r(v_i) = s_i .
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir r(v_1) = nível(v_1) + k

Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

nível(v_1) = 3

Compactar v_1,v_2 e w



Para cada zigue-zague v_1, ..., v_k em H:

- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir r(v_i) = s_i .
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir r(v_1) = nível(v_1) + k

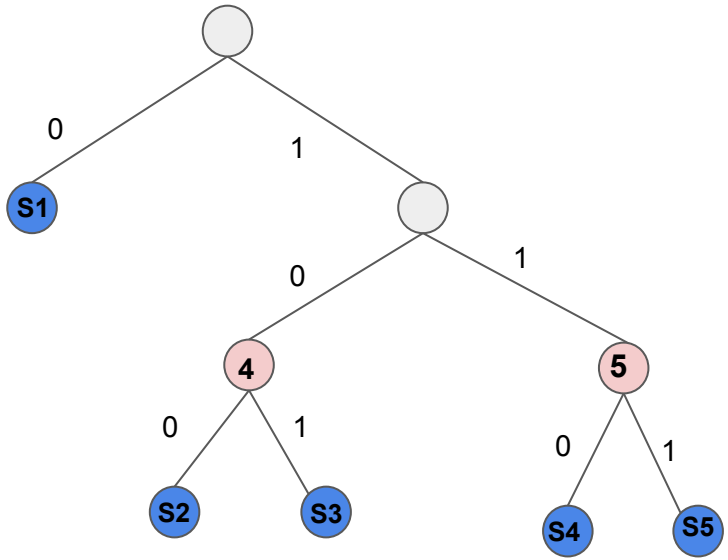
Construção da Árvore Patrícia

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

nível(v_1) = 3

Compactar v_1,v_2 e w

Redefinir r(v_1)

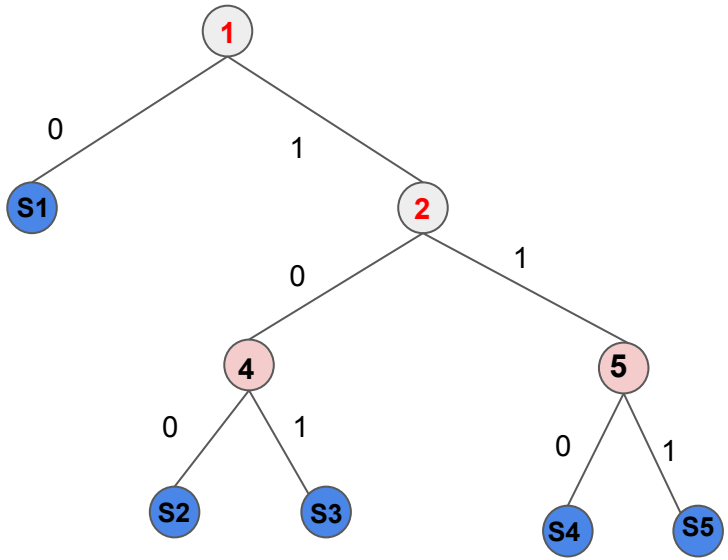


Para cada zigue-zague v_1, ..., v_k em H:

- Se v_k é um nó folha referente à chave s_i , compactar v_1, ..., v_k em v_1 e definir r(v_i) = s_i .
- Caso contrário, v_k possui o filho w. Compactar v_1, ..., v_k, w em v_1 e definir r(v_1) = nível(v_1) + k

Construção da Árvore Patrícia

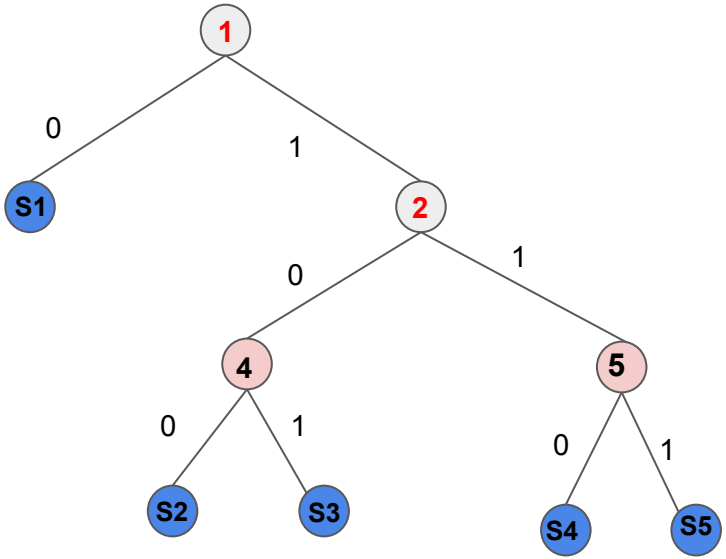
Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101



Se algum vértice v de T permaneceu sem rótulo, definir rótulo $r(v) = nível_H(v)$

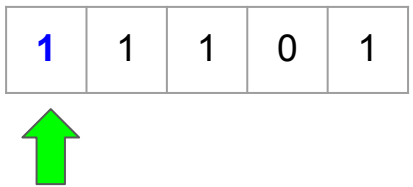
Importante!!!

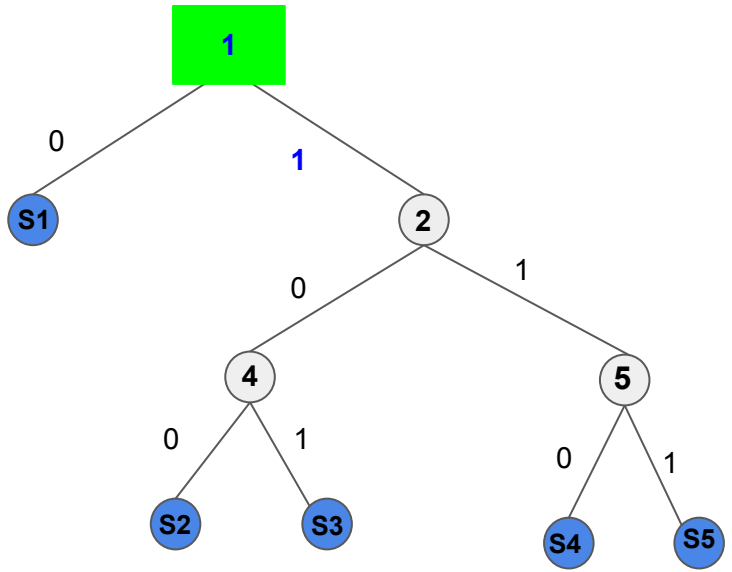
Em uma busca por chave x, o rótulo de um nó v, não folha, é o índice do dígito de x relativo a v



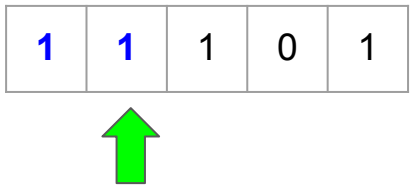
Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

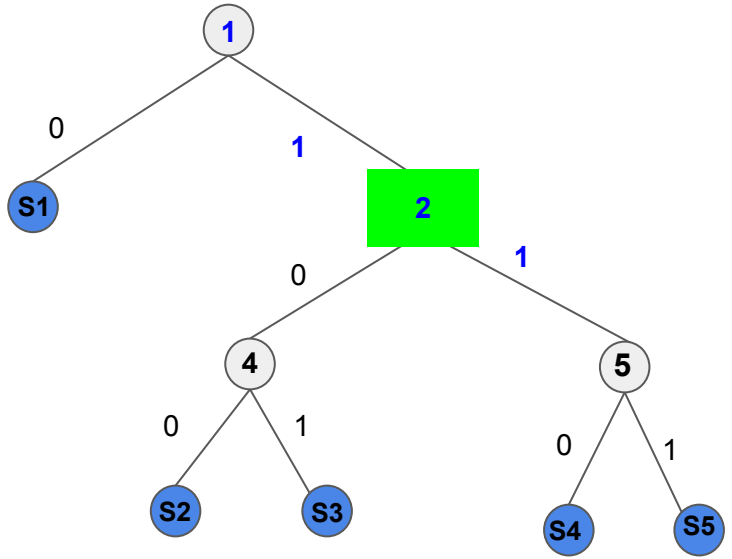
Exemplo: Buscar a chave 11101



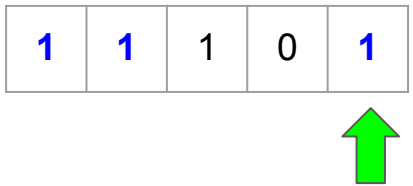


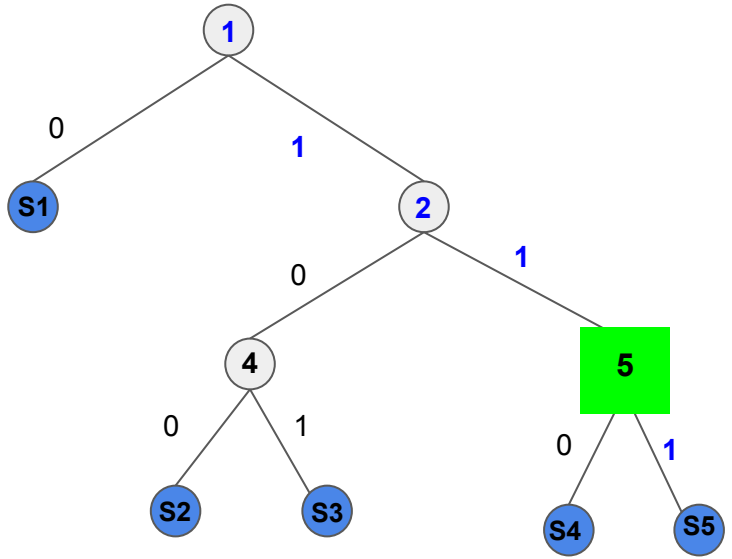
Exemplo: Buscar a chave 11101



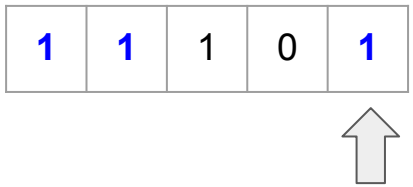


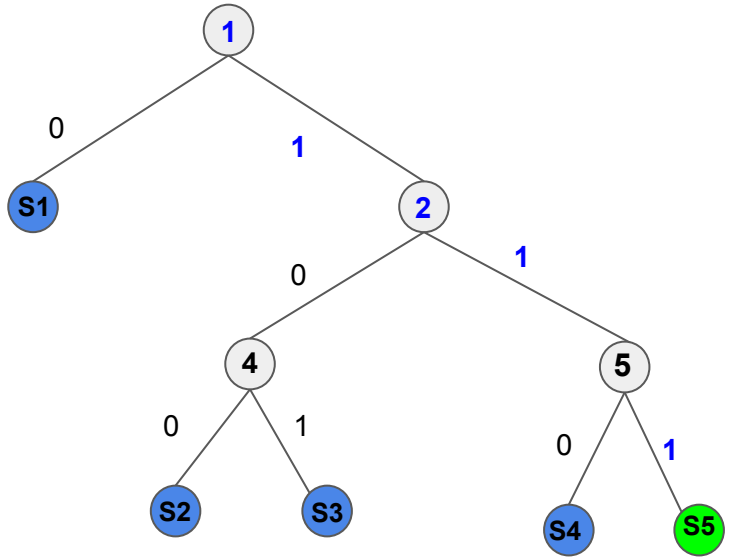
Exemplo: Buscar a chave 11101





Exemplo: Buscar a chave 11101





Busca de uma chave x em T

- Seja *x* a sequência binária $d_1 d_2 ... d_k$ que se deseja buscar
- A partir da raiz de uma árvore Patricia T.
- Supõe que o nó *v* de *T* tenha sido alcançado
  - *v* é folha
    - Se *x* = *r(v)*, a chave foi localizada
    - Caso contrário, *x* é inválida
  - v não é folha
    - Testar *r(v)*-ésimo dígito de *x*, alcançando o filho esquerdo ou direito, se o mesmo for 0 ou 1 , respectivamente.
    - Se *k* < *r(v)*, *x* não corresponde a uma chave, busca termina em um nó interno.

Inserção em uma Árvore Patricia T

Inserção de uma chave *x* composta com *k* dígitos binários *d(1),...,d(k)*

- Efetua-se uma busca da chave *x* em *T*
  - A busca termina em algum nó y (interno ou folha) de T
- Selecione *y'*, um dos nós folha descendente de y.
  - Seja s_i a chave contida em *y'* (Se *y* é folha, *y'* = *y*)
- Seja L o comprimento do maior prefixo comum de *x* e s_i (ou seja, *x* e s_i coincidem exatamente até o *L-ésimo* dígito)
- Seja c, o comprimento de s_i
- Se L = *k*, inserção inválida (x é prefixo de s_i)
- Se L = *c*, inserção inválida (s_i é prefixo de x)

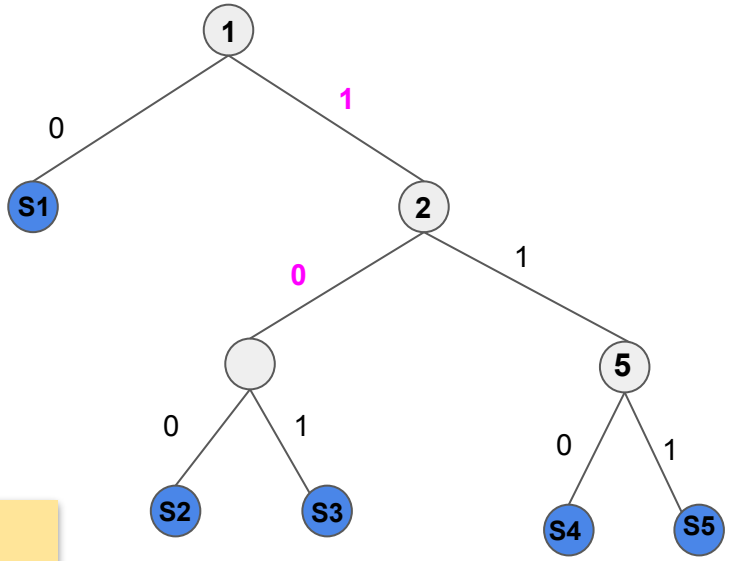
Se inserção for válida

- Determinar nó *z* de *T* onde será realizada a inclusão
- Se *y'* é o único nó em T, então *z* = *y'*
- Senão, seja *z'* o pai de *y'*, e *A* o caminho da raiz de T até *z'*
- Se r(*z'*) ≤ L + 1 então *z* = *y'*
- Quando *r(z')* > *L* + *1*, z será o nó de A mais próximo da raiz de T, tal que *r(z)* > *L* + *1*
- Criar dois nós novos, *v* e *w*, com rótulos *r(v)* = *L* + *1*, *r(w)* = *x*.
  - O pai de v será o antigo pai de z.
  - Os filhos de v serão w e z, sendo w, o filho esquerdo se d(L + 1) = 0 ou o direito quando d(L+ 1) = 1
- Se *z* era a raiz de T, a nova raiz passa a ser v.

Exemplo: Inserção Inválida

- Inserir chave 10

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

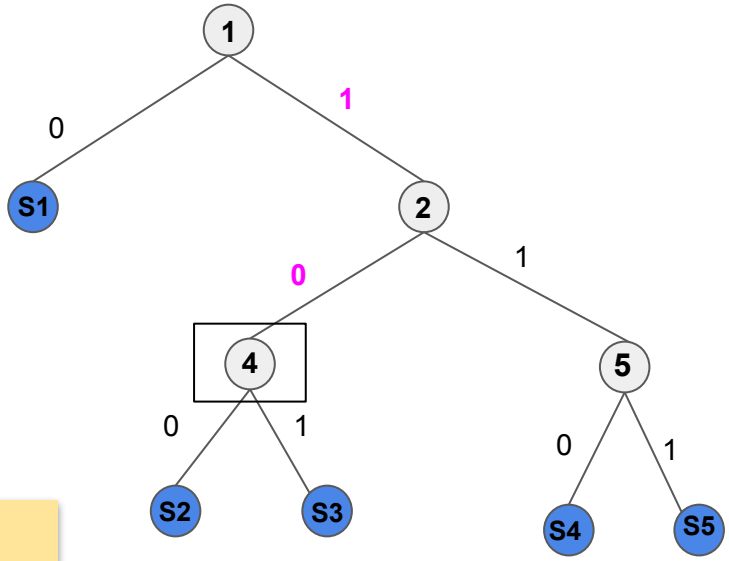


- Efetua-se uma busca da chave *10* em *T*

Exemplo: Inserção Inválida

- Inserir chave 10

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

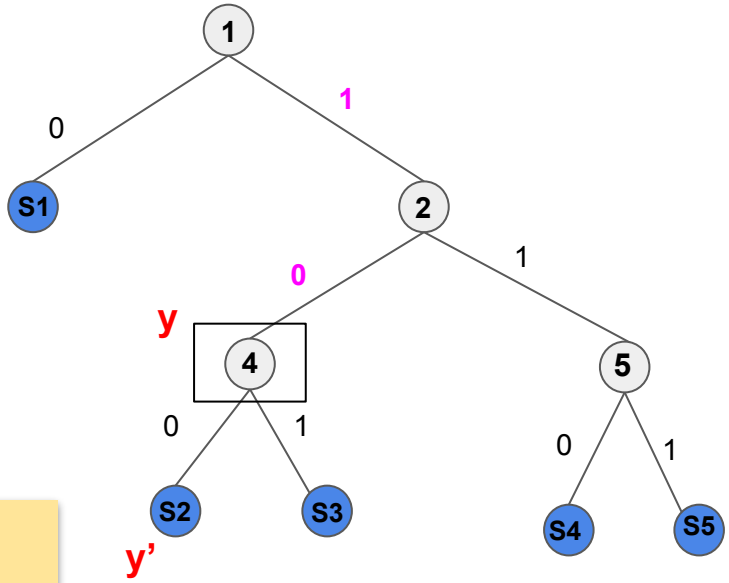


- Efetua-se uma busca da chave *10* em *T*
  - A busca termina em algum nó y interno de T

Exemplo: Inserção Inválida

- Inserir chave 10

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

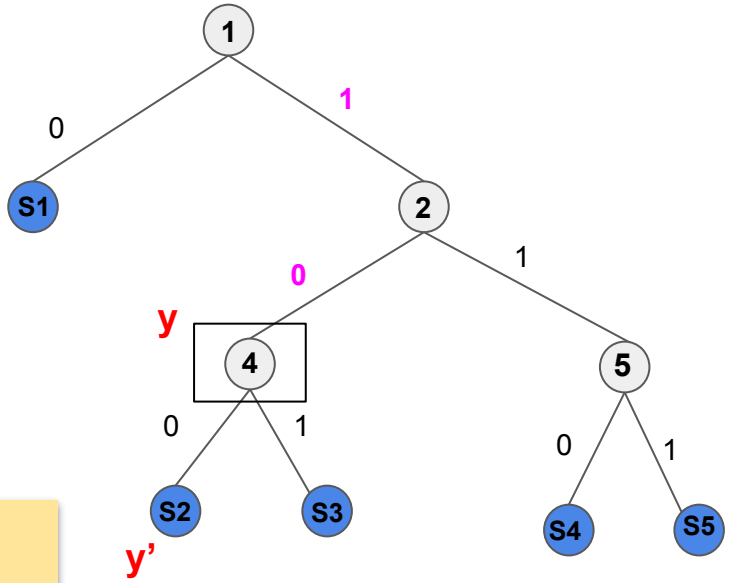


- Efetua-se uma busca da chave *10* em *T*
  - A busca termina em algum nó y interno de T
  - Selecione y', um dos nós folha descendente de y.

Exemplo: Inserção Inválida

- Inserir chave 10

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

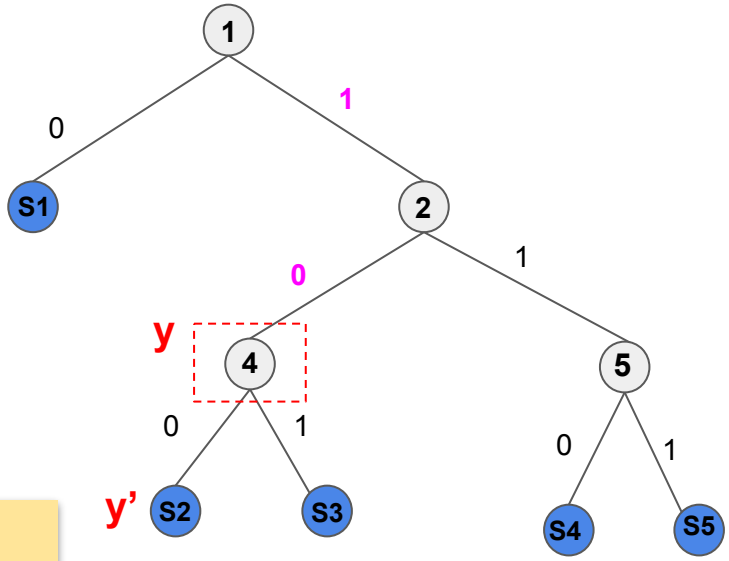


- Efetua-se uma busca da chave *10* em *T*
  - A busca termina em algum nó y interno de T
  - Selecione y', um dos nós folha descendente de y.
    - S2 a chave contida em y'

Exemplo: Inserção Inválida

- Inserir chave 10

Chaves Indexadas:
S1 - 0
S2 - **10**00
S3 - 10010
S4 - 11100
S5 - 11101



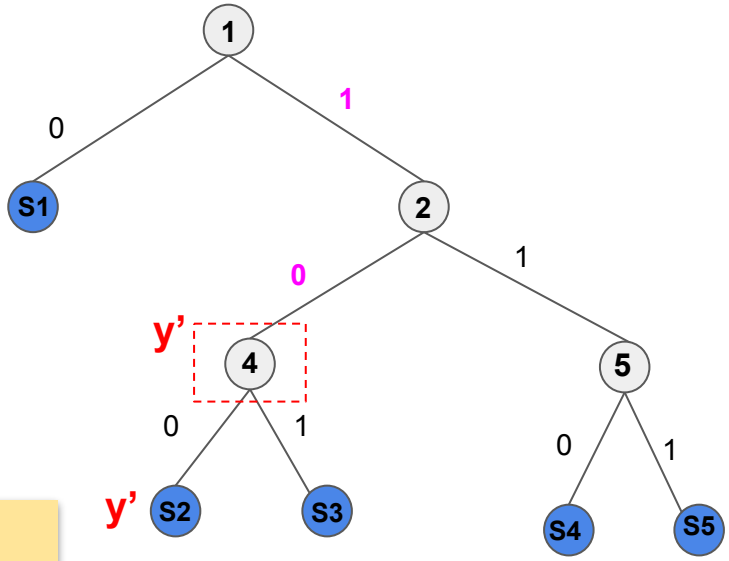
- Seja L o comprimento do maior prefixo comum de *x* e s_i (ou seja, *x* e s_i coincidem exatamente até o *L-ésimo* dígito)
- Seja c, o comprimento de s_i
- Se L == *k*, inserção inválida (x é prefixo de s_i)
- Se L == *c*, inserção inválida (s_i é prefixo de x)

L = 2
k = 2
c = 4

Exemplo: Inserção Inválida

- Inserir chave 10

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101



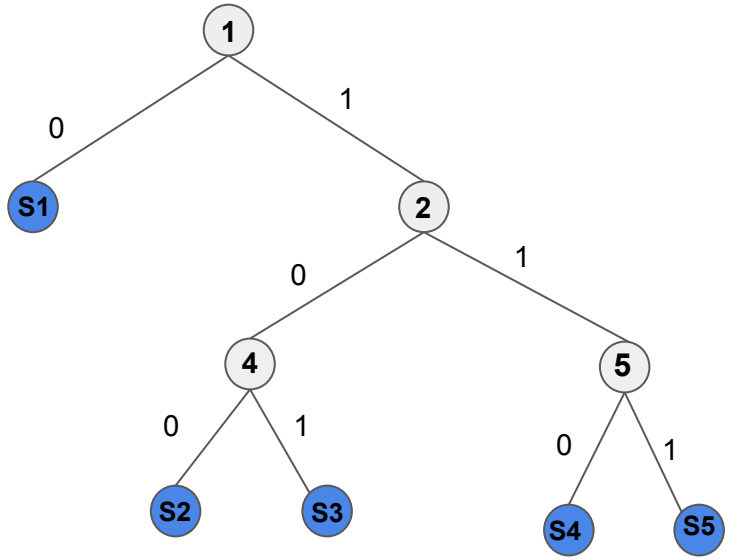
- Seja L o comprimento do maior prefixo comum de *x* e s_i (ou seja, *x* e s_i coincidem exatamente até o *L-ésimo* dígito)
- Seja c, o comprimento de s_i
- **Se L == *k*, inserção inválida (x é prefixo de s_i)**
- Se L == *c*, inserção inválida (s_i é prefixo de x)

L = 2
k = 2
c = 4

Exemplo: Inserção válida

- Inserir chave 111101

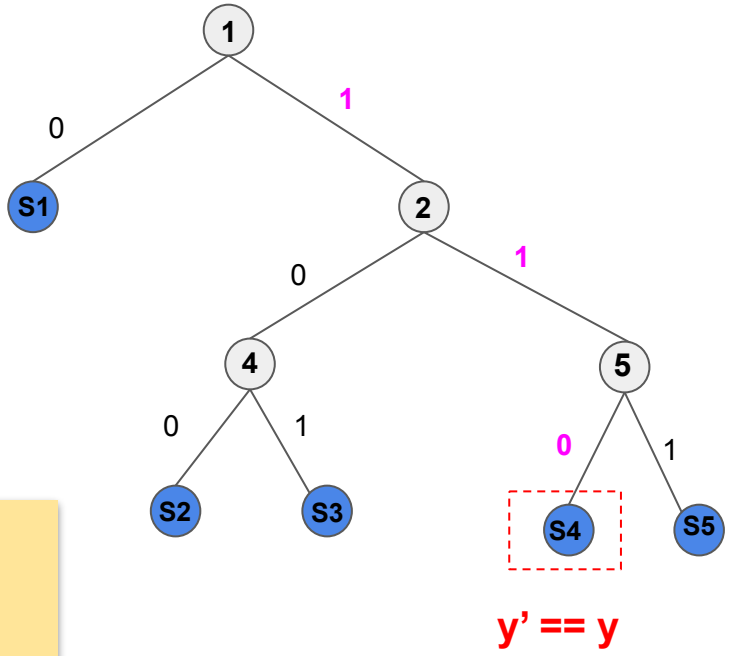
Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101



Exemplo: Inserção válida

- Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

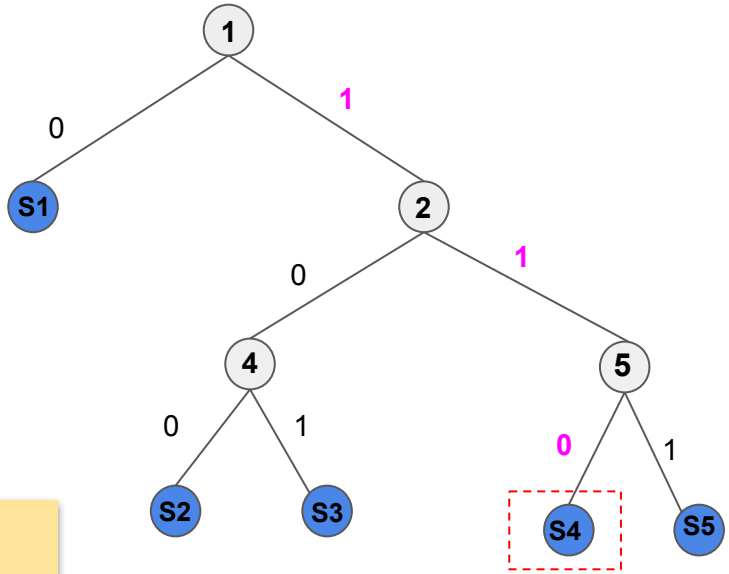


- Efetua-se uma busca da chave *111101* em *T*
  - A busca termina em nó y folha de T
  - Seja s_i a chave contida em y' (Se y é folha, y' = y)

Exemplo: Inserção válida

➢ Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101



- Seja L o comprimento do maior prefixo comum de ***x*** e s_i (ou seja, *x* e s_i coincidem exatamente até o *L-ésimo* dígito)
- Seja c, o comprimento de s_i
- Se L == *k*, inserção inválida (x é prefixo de s_i)
- Se L == *c*, inserção inválida (s_i é prefixo de x)

L = 3
k = 6
c = 5

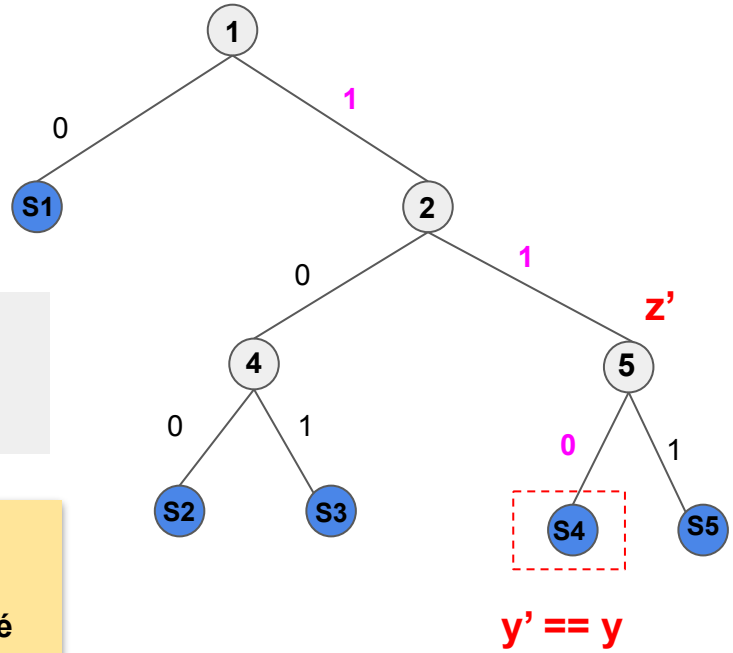
y' == y

Exemplo: Inserção válida

➢ Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

L = 3
k = 6
c = 5



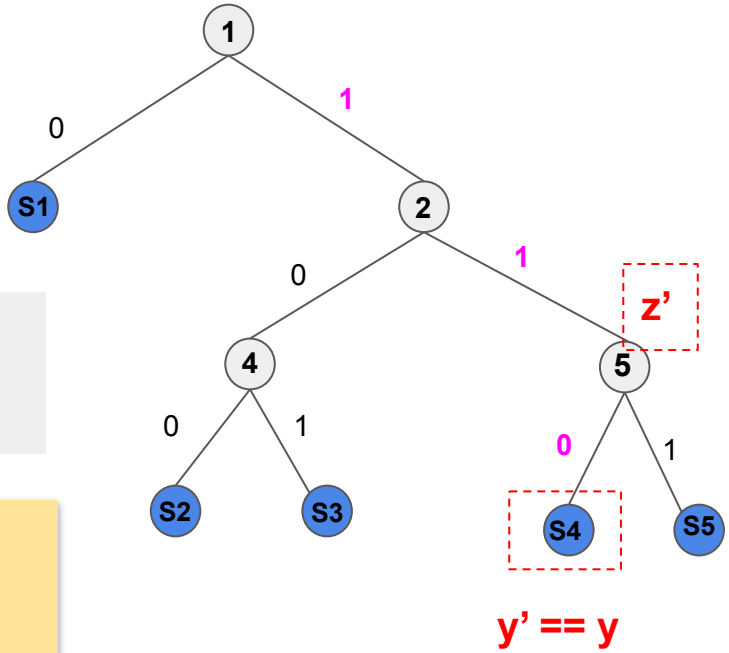
- Determinar nó *z* de *T* onde será realizada a inclusão
  - Se *y'* é o único nó em T, então *z = y'*
  - **Senão, seja *z'* o pai de *y'*, e *A* o caminho da raiz de T até *z'***
- Se r(*z'*) ≤ L + 1 então *z = y'*
- Quando *r(z') > L + 1*, z será o nó de A mais próximo da raiz de T, tal que *r(z) > L + 1*

Exemplo: Inserção válida

➢ Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

L = 3
k = 6
c = 5



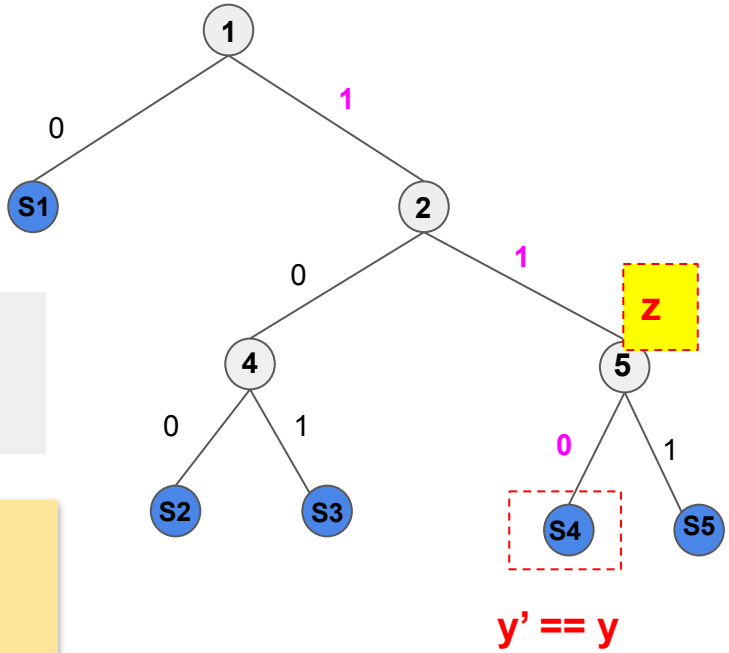
- Determinar nó *z* de *T* onde será realizada a inclusão
  - Se *y'* é o único nó em T, então *z* = *y'*
  - Senão, seja *z'* o pai de *y'*, e *A* o caminho da raiz de T até *z'*
- Se r(*z'*) ≤ L + 1 então *z* = *y'*
- **Quando *r(z')* > *L* + *1*, z será o nó de A mais próximo da raiz de T, tal que *r(z)* > *L* + *1***

Exemplo: Inserção válida

➢ Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

L = 3
k = 6
c = 5



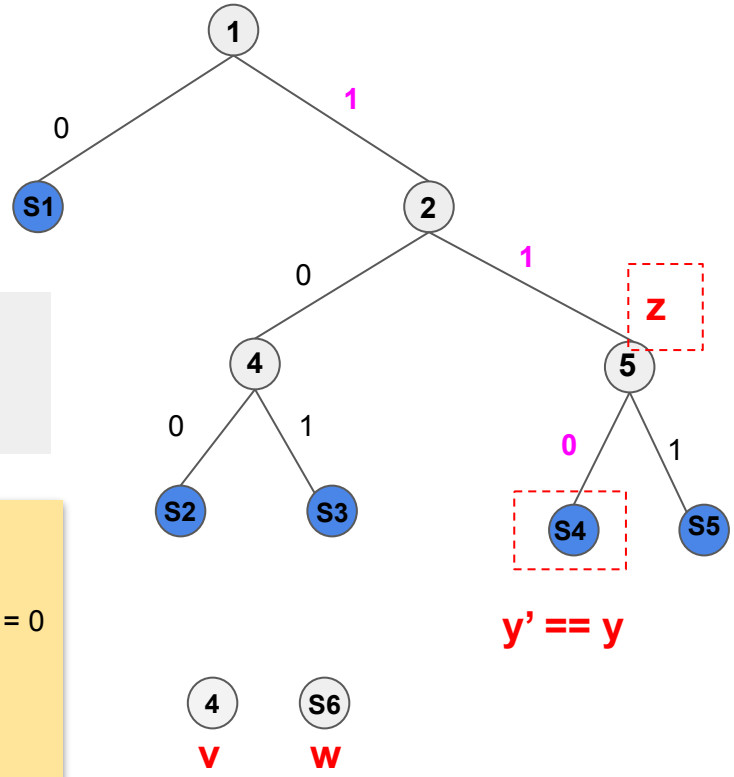
- Determinar nó *z* de *T* onde será realizada a inclusão
  - Se *y'* é o único nó em T, então *z* = *y'*
  - Senão, seja *z'* o pai de *y'*, e *A* o caminho da raiz de T até *z'*
- Se r(*z'*) ≤ L + 1 então *z* = *y'*
- **Quando *r(z')* > *L* + *1*, z será o nó de A mais próximo da raiz de T, tal que *r(z)* > *L* + *1***

Exemplo: Inserção válida

➢ Inserir chave 111101 = S6

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

L = 3
k = 6
c = 5



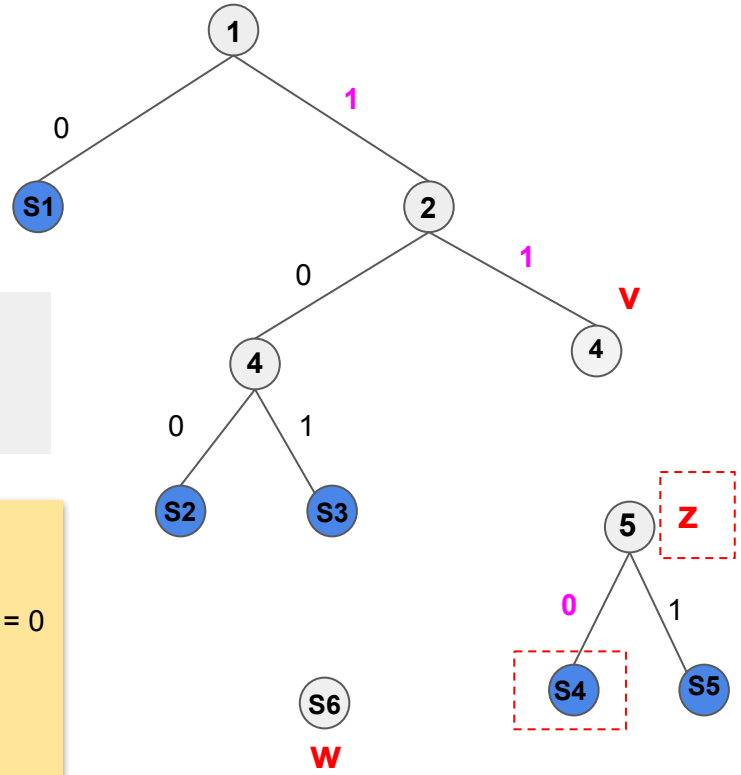
➢ **Criar dois nós novos, *v* e *w*, com rótulos *r(v) = L + 1*, *r(w) = x*.**
- O pai de v será o antigo pai de z.
- Os filhos de v serão w e z, sendo w, o filho esquerdo se d(L + 1) = 0 ou o direito quando d(l + 1) = 1

Exemplo: Inserção válida

➢ Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

L = 3
k = 6
c = 5



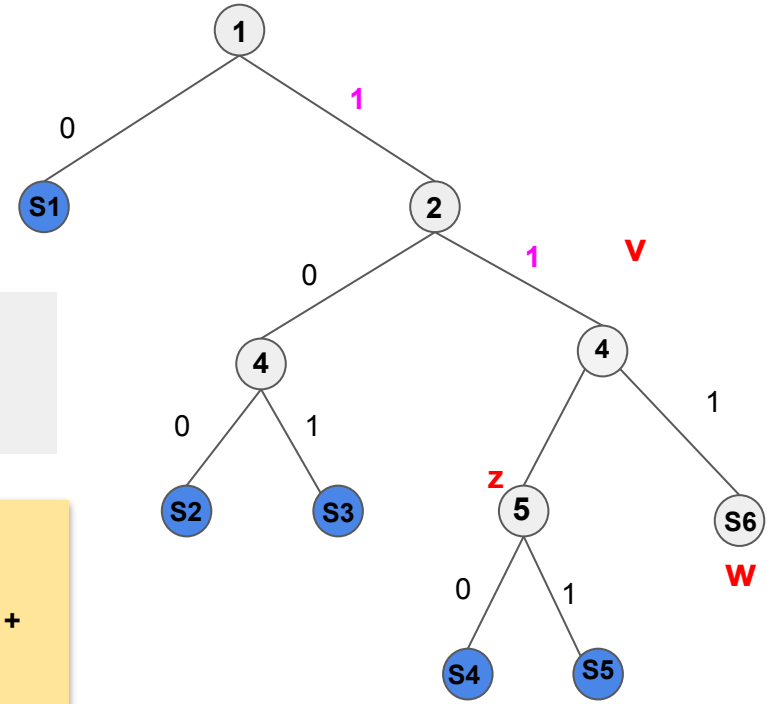
➢ Criar dois nós novos, *v* e *w*, com rótulos *r(v) = L + 1*, *r(w) = x*.
- **O pai de v será o antigo pai de z.**
- Os filhos de v serão w e z, sendo w, o filho esquerdo se d(L + 1) = 0 ou o direito quando d(l + 1) = 1

Exemplo: Inserção válida

- Inserir chave 111101

Chaves Indexadas:
S1 - 0
S2 - 1000
S3 - 10010
S4 - 11100
S5 - 11101

L = 3
k = 6
c = 5

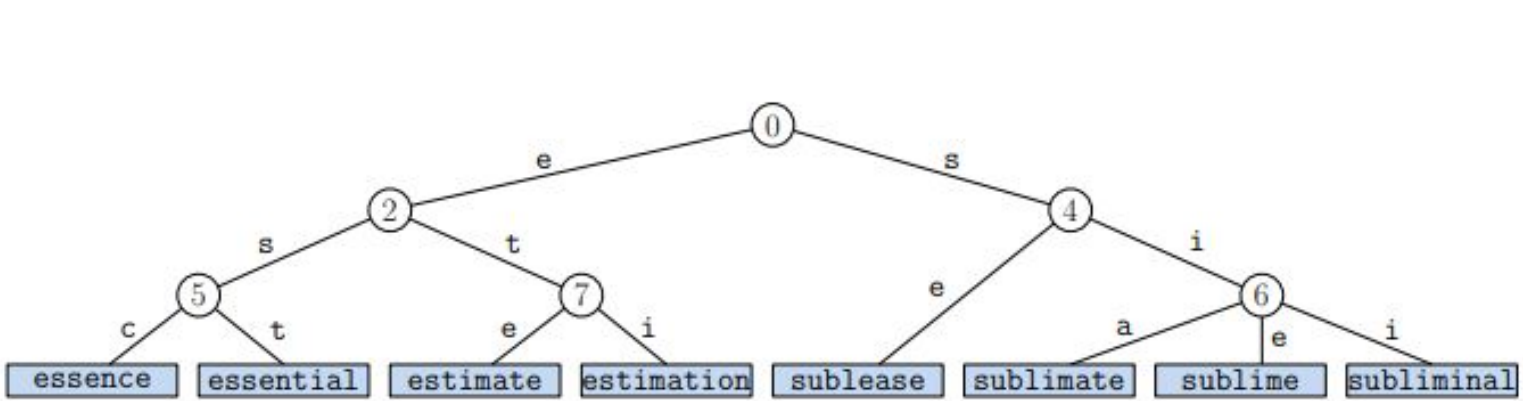


- Criar dois nós novos, *v* e *w*, com rótulos *r(v) = L + 1*, *r(w) = x*.
  - O pai de v será o antigo pai de z.
  - **Os filhos de v serão w e z, sendo w, o filho esquerdo se d(L + 1) = 0 ou o direito quando d(L + 1) = 1**

Propriedade Importante

- Percorrendo-se as folhas da esquerda para a direita, obtém-se as chaves em ordem lexicográfica crescente

Árvore Patricia para um conjunto de strings



0123456789...
essence
essential
estimate
estimation
sublease
sublimate
sublime
subliminal

Exercícios

1. Mostre a trie binária resultante da inserção das chaves `00100 00001 10110 11100 11101`
2. Mostre a árvore patricia resultante da inserção das chaves `00100 00001 10110 11100 11101`

Visualização da Trie

Visualização da Trie comprimida

Referência:

- Szwarcfiter, J.; Markezon, L. Estruturas de Dados e seus Algoritmos, 3a. ed. LTC. Capítulo 11
- Drosdek, A; Estruturas de Dados e Algoritmos em C++. Tradução 4a Edição, CENGAGE Learning. Seção 7.2.
- Donald R. Morrison. 1968. PATRICIA—Practical Algorithm To Retrieve Information Coded in Alphanumeric. J. ACM 15, 4 (Oct. 1968), 514–534. https://doi.org/10.1145/321479.321481
- Material dos Professores Jairo Francisco de Souza (UFJF) e Vanessa Braganholo (UFF)